FORTALESA: Fault-Tolerant Reconfigurable Systolic Array for DNN Inference

Natalia Cherezova*, Artur Jutman*†, Maksim Jenihhin*
*Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia
†Testonica Lab, Tallinn, Estonia
{natalia.cherezova, maksim.jenihhin}@taltech.ee, artur@testonica.com

Abstract—The emergence of Deep Neural Networks (DNNs) in mission- and safety-critical applications brings their reliability to the front. High performance demands of DNNs require the use of specialized hardware accelerators. Systolic array architecture is widely used in DNN accelerators due to its parallelism and regular structure. This work presents a run-time reconfigurable systolic array architecture with three execution modes and four implementation options. All four implementations are evaluated in terms of resource utilization, throughput, and fault tolerance improvement. The proposed architecture is used for reliability enhancement of DNN inference on systolic array through heterogeneous mapping of different network layers to different execution modes. The approach is supported by a novel reliability assessment method based on fault propagation analysis. It is used for the exploration of the appropriate execution mode-layer mapping for DNN inference. The proposed architecture efficiently protects registers and MAC units of systolic array PEs from transient and permanent faults. The reconfigurability feature enables a speedup of up to $3\times$, depending on layer vulnerability. Furthermore, it requires $6 \times$ less resources compared to static redundancy and $2.5 \times$ less resources compared to the previously proposed solution for transient faults.

Index Terms—systolic array, fault tolerance, deep neural networks, accelerator.

I. INTRODUCTION

Recent years have been marked by a tremendous increase in adopting Machine Learning (ML) algorithms, specifically Deep Neural Networks (DNNs), and their employment in mission- and safety-critical application domains, such as automotive, avionics, and space. The computational complexity of DNN models requires the use of specialized hardware accelerators. Many DNN accelerators are based on systolic array architecture, including Google TPU [1], MIT Eyeriss [2], and Berkeley Gemmini [3]. However, as nanoelectronics used in hardware accelerators are prone to faults caused by radiation and aging, the concern about their reliability gets to the front [4].

Furthermore, with the current need for self-aware computing systems that can monitor their state and adapt to changing environments and operating conditions, an important property of hardware design is the ability to adjust or reconfigure to changing conditions for improved availability. To address those needs, we propose a run-time reconfigurable systolic array architecture that tackles the balance between reliability and performance through different execution modes. As a target application, we consider the inference of DNNs. Since different DNN components have different vulnerability levels,

they might benefit from heterogeneous mapping to different execution modes of the proposed systolic array. We also introduce a reliability assessment method to efficiently validate the effectiveness of the proposed architecture and determine the appropriate execution mode—layer mapping.

The contributions of the paper are

- a systolic array architecture with three execution modes (FORTALESA) for run-time reconfigurable redundancy and flexible reliability vs. performance trade-off;
- hardware implementation and evaluation of the proposed architecture;
- a novel reliability assessment method for DNN inference on systolic array based on fault propagation analysis;
- a flexible DNN protection methodology balancing reliability and performance when executed on a systolic array.

For DNN protection, different network layers are mapped to appropriate execution modes of the systolic array based on their vulnerability. The proposed reliability assessment method is used for the exploration of the execution mode—layer mapping.

The proposed architecture efficiently protects both sequential (registers) and combinational (MAC) components of systolic array processing elements. While the main objective is to address transient faults (soft errors), the proposed architecture can handle permanent faults as well allowing for graceful degradation. The reconfigurability feature of FORTALESA enables a speedup up to $3\times$, depending on layer vulnerability, and requires $6\times$ less resources compared to static redundancy and $2.5\times$ less resources than the previously proposed solution for transient faults.

The rest of the paper is organized the following way. Section II gives an overview of related works. Section III presents the background on systolic array architecture. Section IV introduces the architecture of the proposed reconfigurable systolic array. Section V explains the proposed reliability assessment method based on fault propagation. Section VI evaluates the effectiveness of the proposed methodology for DNN reliability, and Section VII concludes the paper.

II. RELATED WORKS

A reconfigurable redundancy approach is common in multicore processors, e.g., ARM's Triple Core Lock-Step (TCLS) system [5] and RISC-V based multicore processing cluster with on-demand redundancy for efficient performance vs. reliability trade-off [6]. Such solutions allow the use of individual

cores separately to run different tasks for high performance or as a group to run the same task for high reliability. Execution mode can be reconfigured during run-time, thus allowing the computing system to adapt to changing conditions and task requirements.

While there are several ways to ensure the reliability of DNN inference, the most common approaches imply redundancy, fault-aware training, and algorithm-based fault tolerance (ABFT). Redundancy indicates replication of the components working in parallel (spatial redundancy) or reexecution of the same operation (temporal redundancy). Since redundancy incurs high overhead, the goal is to find a trade-off between reliability and performance by identifying and protecting more vulnerable components. To tackle this challenge, prior works have proposed selective replication of the layers [7], [8], of the channels within a layer [9], of the weights [10] or neurons [11]. Another common approach requires retraining of the DNN with the added faults to improve the resilience of the network. Fault-aware training was used to address faults in FPGAs [12], timing [13] and computational [14] errors in accelerators. ABFT methods consider certain features of the application for fault detection and correction. Proposed solutions include checksums [15], [16], bounded activation functions [17], restricted output ranges [18] and quantile shifts [19], and AN codes [20].

Systolic arrays were first introduced in the 1980s [21] and recently became popular for accelerating DNNs. As the reliability of DNN accelerators comes to the front, recent works have proposed methods for the assessment and enhancement of systolic arrays. [22] analyzed the effects of transient faults in the combinational logic of systolic arrays. [23] presented Saca-FI, a microarchitecture-level fault injection framework for systolic array-based CNN accelerators, for the analysis of transient faults in registers. Saca-FI is based on SCALE-Sim, a cycle-accurate simulator for DNN inference on systolic arrays [24]. [25] proposed an RTL-level fault injection framework for the analysis of stuck-at faults within multiply and accumulate (MAC) units. [26] presented a fault injection framework that models systolic array dataflow using Uniform Recurrent Equations. Unlike previous works, the proposed reliability assessment method is based on fault propagation analysis. Instead of injecting faults in the microarchitectural or RTL model of a systolic array, which is a very time-consuming task; errors are added directly to the DNN layer output using fault propagation analysis.

Reliability enhancement methods for systolic arrays, similarly to DNNs, include ABFT and network retraining. [27] presented a permanent fault correction technique. [28] proposed fault-aware pruning plus retraining (FAP+T) technique to address manufacturing defects in the systolic array-based accelerators. It requires modification of the target DNN architecture through pruning and retraining to adapt it to a specific defect in the chip. [29] presented an ABFT technique for permanent fault detection in systolic arrays on FPGAs. Permanent faults in configuration memory are considered in that work. Another permanent fault detection approach assuming faults in registers and MAC units is presented in [30]. The approach is based on checksum calculations and was implemented on

FPGA as well. While most works address permanent faults, [31] proposed an ABFT method for detecting timing faults due to reduced voltage. Compared to the aforementioned works, the proposed reconfigurable systolic array architecture addresses both permanent and transient faults and does not require retraining of a DNN for each particular chip.

Reconfigurable DNN accelerators based on a systolic array have been proposed before: to improve energy efficiency [32], to decrease inference latency [33] and power consumption [34], to boost transformer models performance [35]. In contrast to the previous works, the proposed reconfigurable systolic array architecture improves reliability of the DNN inference while balancing performance.

III. BACKGROUND

A. Systolic array architecture

Systolic array has a regular structure consisting of an array of Processing Elements (PEs). Each PE consists of a Multiply-Accumulate (MAC) unit and registers to hold the intermediate values. Memory buffers for activations and weights are located at the top and left sides of the systolic array correspondingly. Commonly, weights are fed vertically from top to bottom, thus each column is assigned to one output channel, and activations (inputs) are fed from left to right, hence each row processes values from the same sliding windows. Dataflow type defines the internal structure of PEs and the movement of data through the array. Dataflow types include weight-stationary (WS), input-stationary (IS), and output-stationary (OS) [36].

In the **weight-stationary dataflow**, the weight matrix is first pre-loaded into a systolic array by loading one row per cycle, then the input matrix propagates through the array from left to right. The outputs (results of MAC operation) propagate from top to bottom. In this scenario, weights are stationary and are kept in the systolic array during matrix multiplication execution. This dataflow maximizes the reuse of the weights.

In the **input-stationary dataflow**, activations are pre-loaded in the systolic array in a similar manner as WS and kept in the systolic array during the matrix multiplication process. The data movement and structure of the systolic array are similar to WS dataflow.

In the **output-stationary dataflow**, outputs are kept in the PEs of the systolic array, while weights and activations are streaming through the systolic array (Fig. 1). Weights are streaming from top to bottom, and activations (inputs) are streaming from left to right. Outputs are accumulated inside PEs (Fig. 2). This dataflow type does not need a pre-loading stage. An output-stationary systolic array is considered for this work. It is commonly used and allows high inputs and outputs reuse [37], [38] and on average has higher performance [33]. Previous works have shown that it is more fault tolerant [39].

The convolution operation is mapped to the systolic array by transforming it to matrix multiplication using im2col algorithm [40]. An input tensor is transformed into a $(H_{out} \cdot W_{out}) \times (H_k \cdot W_k \cdot C_{out})$ matrix and weights are transformed into a $(H_k \cdot W_k \cdot C_{in}) \times C_{out}$ matrix, where H_{out} and W_{out} are the height and width of the output tensor, H_k and W_k are the

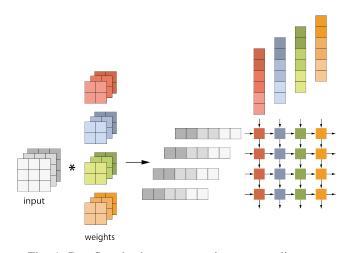


Fig. 1: Dataflow in the output-stationary systolic array

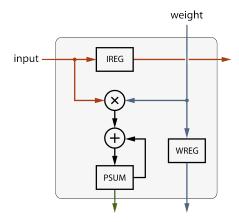


Fig. 2: Architecture of the individual PE in the outputstationary systolic array

height and width of a kernel, C_{in} and C_{out} are the number of channels in the input and output tensor correspondingly. When inputs and weights are represented this way, each column of the output-stationary systolic array calculates values for one output channel, and each row calculates values of the same pixel across all output channels.

If the input matrices exceed the size of the systolic array, matrix multiplication is performed in tiles corresponding to the size of the systolic array grid.

The latency L of a matrix multiplication on a systolic array is calculated as

$$L_{SA} = M + N - 1 + N - 1 = M + 2N - 2,$$
 (1)

where $N \times N$ is the size of the systolic array and $N \times M$ is the size of the matrices [24].

The number of tiles in each matrix is calculated as

$$T_a = \left\lceil \frac{P}{N} \right\rceil \,, \tag{2}$$

$$T_{a} = \left\lceil \frac{P}{N} \right\rceil , \qquad (2)$$

$$T_{w} = \left\lceil \frac{K}{N} \right\rceil , \qquad (3)$$

where T_a and T_w are the number of tiles in activations and weights correspondingly, P is the size of the reshaped activations, and K is the number of output channels and the size of the weights matrix. The total amount of steps needed to calculate the output equals $S = T_a \cdot T_w$, and the total latency of matrix multiplication on the systolic array is therefore

$$L_{total} = S \cdot L_{SA} = \left\lceil \frac{P}{N} \right\rceil \cdot \left\lceil \frac{K}{N} \right\rceil \cdot (M + 2N - 2). \tag{4}$$

B. Fault propagation in systolic arrays

Considering the structure of a single processing element, faults can occur in registers holding intermediate values or in a multiplier. Considering the regular structure of a systolic array and data movement, faults in different components produce different error patterns in the output. Fig. 3 presents the possible patterns for the output-stationary systolic array. A bit flip in a weight register (WREG) results in the propagation of faulty weight through the whole column affecting several output values in one channel. This ends up in the *line pattern*. A bit flip in an input register (IREG) results in the propagation of faulty activation through the whole row affecting one value in several output channels. This ends up in the bullet pattern. A bit flip in the multiplier or an output register keeping partial sum (OREG) affects one value in one output channel resulting in a point pattern.

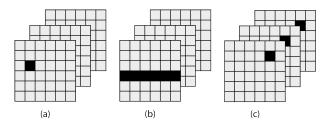


Fig. 3: Error patterns in the output tensor: (a) single point, (b) line, (c) bullet

IV. FAULT-TOLERANT RECONFIGURABLE SYSTOLIC ARRAY

We propose FORTALESA, a systolic array with reconfigurable redundancy that provides flexible reliability performance trade-off for critical tasks requiring fault tolerance and non-critical ones that benefit from increased parallelism.

The proposed architecture supports three **execution modes**:

- no redundancy for increased performance;
- dual modular redundancy for reliability performance trade-off;
- triple modular redundancy for full protection.

The selected mode is configured through a control signal that controls multiplexers. The control signal is assumed to come from the host processor based on the task requirement. Since different layers of DNNs have different vulnerability, it is possible to execute different layers in different modes, thus achieving an efficient trade-off between performance and reliability.

DMR execution mode. In the DMR mode, two neighboring PEs form a group (Fig. 4). They receive the same inputs, one of them acts as a main PE, the other as a shadow. Main PE compares two partial sums (OREG) and adjusts its partial sum value without re-execution using one of possible fault

correction techniques: (a) averaging of two values (DMRA) or (b) setting faulty bits to zero (DMR0). The choice of the fault correction technique is a design-time parameter that defines two implementation options for the DMR mode.

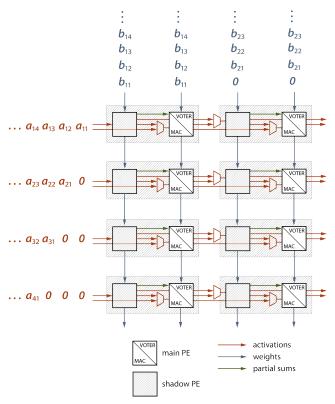


Fig. 4: Arrangement of the systolic array in the dual modular redundancy execution mode (DMRA or DMR0)

Additional connections are introduced to support redundant modes. To ensure that faulty IREG or WREG values do not propagate through the array, each PE passes the activation to the next PE of the same type skipping one in between. The shadow PE also passes the value of the OREG (partial sum) to the main PE for comparison. Main PE has additional functionality for fault correction that works in parallel with the MAC unit. Since the updated partial sum is available on the next clock cycle, the latency of the systolic array working in DMR mode equals the latency of the $N \times N/2$ systolic array plus 1:

$$L_{SA}^{dmr} = M + N - 1 + \frac{N}{2} - 1 + 1 = M + \frac{3N}{2} - 1$$
. (5)

The total latency is then

$$L_{total}^{dmr} = \left\lceil \frac{P}{N} \right\rceil \cdot \left\lceil \frac{2K}{N} \right\rceil \cdot \left(M + \frac{3N}{2} - 1 \right). \tag{6}$$

Since in the redundancy execution modes, several PEs calculate the same values, the term **effective size** of the systolic array is introduced. It defines the number of groups calculating unique values and the size of the output matrix. The effective size of the systolic array in the DMR execution mode is $N \times \frac{N}{2}$.

TMR execution mode. For the TMR execution mode, two implementations are possible: with groups of three or four PEs. The implementation option with groups of three PEs (TMR3)

is shown in Fig. 5 and the implementation option with groups of four PEs (TMR4) in Fig. 6. The group size is the design-time parameter for the TMR mode.

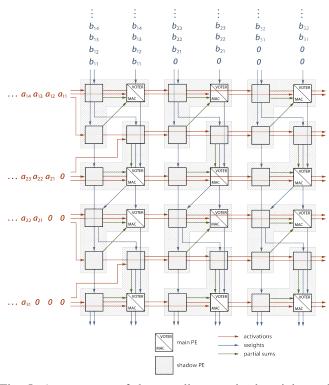


Fig. 5: Arrangement of the systolic array in the triple modular redundancy execution mode, implementation option with groups of three PEs (TMR3)

In the TMR3 implementation, the voter in the main PE works in parallel with the MAC unit. In the TMR4 implementation, the MAC unit in the main PE is not used in the TMR mode, the main PE only compares the partial sums calculated by three shadow PEs.

The effective size of the systolic array, i.e., the size of the output matrix, for TMR3 implementation is $\frac{2N}{3}\times\frac{N}{2}$. The effective sizes of the matrices that can be multiplied are $\frac{2N}{3}\times M$ and $M\times\frac{N}{2}$. The latency of the TMR3 implementation is

$$L_{SA}^{tmr3} = M + \frac{2N}{3} - 1 + \frac{N}{2} - 1 + 1 = M + \frac{7N}{6} - 1$$
. (7)

The total latency then

$$L_{total}^{tmr3} = \left\lceil \frac{3P}{2N} \right\rceil \cdot \left\lceil \frac{2K}{N} \right\rceil \cdot \left(M + \frac{7N}{6} - 1 \right). \tag{8}$$

The effective size of the systolic array for the TMR4 implementation is $\frac{N}{2} \times \frac{N}{2}$. The effective sizes of the matrices that can be multiplied are $\frac{N}{2} \times M$ and $M \times \frac{N}{2}$. The latency of the TMR4 implementation is, therefore,

$$L_{SA}^{tmr4} = M + \frac{N}{2} - 1 + \frac{N}{2} - 1 + 1 = M + N - 1.$$
 (9)

The total latency then

$$L_{total}^{tmr4} = \left\lceil \frac{2P}{N} \right\rceil \cdot \left\lceil \frac{2K}{N} \right\rceil \cdot (M+N-1). \tag{10}$$

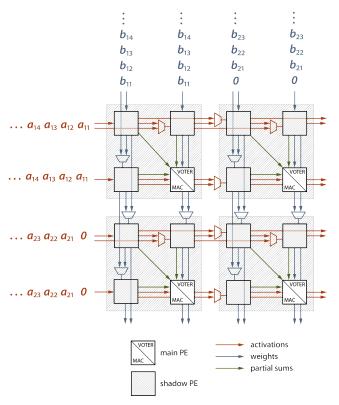


Fig. 6: Arrangement of the systolic array in the triple modular redundancy execution mode, implementation option with groups of four PEs (TMR4)

Since both execution modes have design-time parameters, four implementation options are possible: PM-DMR0-TMR3, PM-DMR0-TMR4, PM-DMRA-TMR3, and PM-DMRA-TMR4. Their efficiency is evaluated in Section VI. The overview of the execution modes and implementation options is given in Table I.

TABLE I: Execution modes and implementation options

Execution mode	Implementation options	Effective size	
Performance (PM)	Baseline SA	$N \times N$	
DMR	DMRA DMR0	$\frac{N}{2} \times N$	
TMR	TMR3	$\frac{2N}{3} \times \frac{N}{2}$	
TWIK	TMR4	$\frac{N}{2} \times \frac{N}{2}$	

V. RELIABILITY ASSESSMENT

This section presents the proposed reliability assessment method. First, for the analysis of transient and permanent faults in the regular OS systolic array used for acceleration of DNNs. Then, for assessment of the proposed architecture since it introduces the DMR execution mode with fault correction. Lastly, it explains the implementation of the assessment method.

The proposed reliability assessment method combines fault injection with fault propagation analysis. Instead of modeling a systolic array on the microarchitecture level, fault propagation analysis is used to calculate the resulting error in the layer output.

A. Fault propagation analysis for transient faults

Transient faults in registers and MAC units are considered for the analysis. Each fault is defined by seven parameters given in Table II.

TABLE II: Transient fault parameters

Symbol	Description
f_{type}	Type of fault (IREG, WREG, OREG, MULT)
ts	Cycle of a systolic array execution
t_w	Weight tile
t_a	Activation tile
p_{row}	Row of the PE
p_{col}	Column of the PE
eta_f	Bit index

Fault in the input register of a systolic array. An output of the convolutional layer can be expressed as

$$y_{(u,v)} = \sum_{c}^{C_{in}} \sum_{i}^{H_k} \sum_{j}^{W_k} w_{(c,i,j)} \cdot x_{(c,u+i,v+j)} + b, \qquad (11)$$

where $y_{(u,v)}$ is the output value with index (u,v), C_{in} is the number of input channels, H_k and W_k are the height and width of a kernel, $w_{(c,i,j)}$ is an individual weight, $x_{(c,u+i,v+j)}$ is an activation (input), and b is a bias.

A bit flip in an input can be expressed as an addition of the error term ε to this input. The error term for a signed integer is expressed as

$$\varepsilon = 2^{\beta_f} \cdot \gamma \,, \tag{12}$$

where β_f is the index of a faulty bit and γ is a sign coefficient, which is derived by the following equation

$$\gamma = \begin{cases}
-1 & \text{if} \quad x_f^{(\beta_f)} = 1 \land \beta_f \neq \beta_{sign} \\
1 & \text{if} \quad x_f^{(\beta_f)} = 1 \land \beta_f = \beta_{sign} \\
-1 & \text{if} \quad x_f^{(\beta_f)} = 0 \land \beta_f = \beta_{sign} \\
1 & \text{if} \quad x_f^{(\beta_f)} = 0 \land \beta_f \neq \beta_{sign}
\end{cases}$$
(13)

where $x_f^{(\beta_f)}$ is the faulty bit of the affected input and β_{sign} is the index of a sign bit.

Then, an error added to the output of the convolutional layer equals

$$e_{ireg} = w_{(c_f, i_f, j_f)} \cdot \varepsilon, \tag{14}$$

where $w_{(c_f,i_f,j_f)}$ is the weight multiplied with the faulty activation. This weight is found by its position (c_f,i_f,j_f) :

$$c_f = \left\lceil \frac{ts - p_{col} - p_{row}}{H_k \cdot W_k} \right\rceil$$

$$k_f = (ts - p_{col} - p_{row}) \mod (H_k \cdot W_k)$$
(15)

$$k_f = (ts - p_{col} - p_{row}) \mod (H_k \cdot W_k)$$
 (16)

$$i_f = \left\lfloor \frac{k_f}{W_k} \right\rfloor \tag{17}$$

$$j_f = k_f \mod W_k \tag{18}$$

where ts is the targeted cycle of a systolic array execution, p_{row} and p_{col} are the position of the targeted PE.

Fault results in the bullet error pattern. Therefore, the following parameters define affected output values: output channels $c_{out,f}$ and position in the feature map (u_f,v_f) . Affected output channels are calculated as follows:

$$c_{out,f} = \left[c_{out,f}^{start}, c_{out,f}^{end}\right] \tag{19}$$

$$c_{out, f}^{start} = (t_w - 1) \cdot N + p_{col} + 1$$
 (20)

$$c_{out,f}^{end} = \begin{cases} t_w \cdot N & \text{if} \quad C_{out} \geqslant t_w \cdot N \\ C_{out} & \text{if} \quad C_{out} < t_w \cdot N \end{cases}$$
 (21)

where $c_{out,f}$ is a range of affected output channels, starting with $c_{out,f}^{start}$ and ending with $c_{out,f}^{end}$, t_w is the targeted weight tile, p_{row} is the row of the targeted PE, and C_{out} is the total number of output channels.

The position (u_f, v_f) is found in the following way:

$$\Box_f = (t_a - 1) \cdot N + p_{row} \tag{22}$$

$$u_f = \left\lfloor \frac{\Box_f}{W_{out}} \right\rfloor \tag{23}$$

$$v_f = \Box_f \mod W_{out} \tag{24}$$

where \Box_f is the affected sliding window and W_{out} is the width of the output feature map.

Fault in the weight register of a systolic array. An error added to the output of the convolutional layer in this case equals

$$e_{wreg} = x_{(c_f, i_f, j_f)} \cdot \varepsilon. \tag{25}$$

The sign of the ε is defined by the affected weight following equation (13).

Fault in the weight register results in the line error pattern, therefore, the following parameters define affected output values: output channel $c_{out,f}$ and a range of positions starting with $(u_f^{start}, v_f^{start})$ and ending with (u_f^{end}, v_f^{end}) . The affected output channel is calculated as follows:

$$c_{out,f} = (t_w - 1) \cdot N + p_{col}$$
. (26)

Positions are calculated from the affected sliding windows following equations (23) and (24):

$$\Box_f^{start} = (t_a - 1) \cdot N + p_{row} \tag{27}$$

$$\Box_f^{end} = \begin{cases} t_a \cdot N & \text{if } W_{out} \cdot H_{out} \geqslant t_a \cdot N \\ W_{out} \cdot H_{out} & \text{if } W_{out} \cdot H_{out} < t_a \cdot N \end{cases}$$
(28)

Faults in the output register and the multiplier. Since those faults result in a single point error pattern, an error added to the output equals

$$e_{oreg} = e_{mult} = 2^{\beta_f} \,. \tag{29}$$

The output channel $c_{out,f}$ is found using equation (26), and the position is found using equations (23) and (24).

B. Fault propagation analysis for permanent faults

Permanent faults are present during the whole execution and therefore defined by four parameters given in Table III.

Permanent faults result in several error patterns, one for each step. Therefore, the affected output values are defined by sets

TABLE III: Permanent fault parameters

Symbol	Description
$f_{type} \ p_{row} \ p_{col} \ eta_f$	Type of fault (IREG, WREG, OREG, MULT) Row of the PE Column of the PE Bit index

of parameters. E.g., in case of permanent faults in the input registers, the affected output values are defined by a set of output channels

$$\left\{ (c_{out,f}^{start}, c_{out,f}^{end})_i \mid i \in \{1, ..., T_w\} \right\}$$
 (30)

and a set of positions

$$\{(u_f, v_f)_i \mid i \in \{1, ..., T_a\}\}.$$
 (31)

A set of output channels is found by substituting t_w with an iterator in equations (20) and (21):

$$c_{out,f(i)}^{start} = (i-1) \cdot N + p_{col} + 1 \ \forall \ i \in \{1, ..., T_w\}$$
 (32)

$$c_{out,f(i)}^{end} = \begin{cases} i \cdot N & \text{if} \quad C_{out} \geqslant i \cdot N \\ C_{out} & \text{if} \quad C_{out} < i \cdot N \end{cases} \quad \forall \ i \in \{1, ..., T_w\}$$
(33)

which gives a pair $(c_{out,f}^{start}, c_{out,f}^{end})$ for every weight tile.

A set of positions is found by substituting t_a with an iterator in equation (22) for the affected sliding windows and then using equations (23) and (24) to calculate position from the sliding window:

$$\Box_{f(i)} = (i-1) \cdot N + p_{row} \ \forall \ i \in \{1, ..., T_a\}$$
 (34)

$$u_{f(i)} = \left| \frac{\square_{f(i)}}{W_{out}} \right| \tag{35}$$

$$v_{f(i)} = \square_{f(i)} \mod W_{out} \tag{36}$$

which gives a position for every input tile. This way, every bullet pattern is characterized by a Cartesian product of those two sets.

An error added to the output for each bullet pattern is the cumulative value

$$e_{ireg}^{p} = \sum_{c}^{C_{in}} \sum_{i}^{H_{k}} \sum_{j}^{W_{k}} w_{(c,i,j)} \cdot \varepsilon_{(c,u_{f}+i,v_{f}+j)}$$
 (37)

where $\varepsilon_{(c,u_f+i,v_f+j)}$ is the error term added to input $x_{(c,u_f+i,v_f+j)}$. Since permanent faults are modeled as stuckat-0 or stuck-at-1 faults, an error term is expressed as

$$\varepsilon = \begin{cases} 2^{\beta_f} \cdot \gamma & \text{if } x_f^{(\beta_f)} \neq s \\ 0 & \text{if } x_f^{(\beta_f)} = s \end{cases}$$
 (38)

where s is the stuck-at state (0 or 1).

C. Assessment of the proposed architecture

Faults in the DMR mode with averaging. The effect of the fault correction on the error added to the output is defined by the affected PE, whether it is a main PE or a shadow. If

the fault happens in the main PE, then fault correction works the following way:

$$y_1^{corr} = \frac{y_1 + e + y_1}{2} = y_1 + \frac{e}{2} = y_1 + \frac{e}{2^1}$$

$$y_2^{corr} = \frac{y_2 + e/2 + y_2}{2} = y_2 + \frac{e}{4} = y_2 + \frac{e}{2^2}$$
...
$$y_n^{corr} = \frac{y_n + e/2^{n-1} + y_2}{2} = y_n + \frac{e}{2^n}$$
(39)

where y_n is the partial sum on the n-th cycle after the fault, y^{corr} is the corrected partial sum used by the main PE. As the number of clock cycles after the fault increases, the value of the added error decreases approaching zero.

If the fault happens in the shadow PE, then fault correction works this way:

$$y_1^{corr} = \frac{y_1 + y_1 + e}{2} = y_1 + \frac{e}{2} = y_1 + \frac{e}{2^1}$$

$$y_2^{corr} = \frac{y_2 + e/2 + y_2 + e}{2} = y_2 + \frac{3e}{4} = y_2 + \frac{(2^2 - 1)e}{2^2}$$
...
$$y_n^{corr} = \frac{y_n + (2^{n-1} - 1)e/2^{n-1} + y_2 + e}{2}$$

$$= y_n + \frac{(2^n - 1)e}{2^n}$$
(40)

As the number of clock cycles after the fault increases, the value of the added error approaches itself.

Faults in the DMR mode with setting mismatched bits to zero. Since the effect of setting mismatched bits to zero cannot be represented as an addition of an error term, values of affected outputs are re-calculated completely. Algorithm 1 presents an algorithm for calculating the effect of faults in the input register assuming the fault happens in the main PE.

D. Implementation

The workflow of the fault injection is presented in Fig. 7. Pytorch is used to run the inference of the network, leveraging GPU speed up. The fault injection module takes the output of the targeted layer and modifies only affected output values according to the aforementioned formulas. The simulation then continues with the erroneous values.

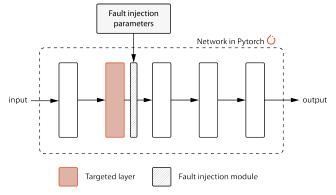


Fig. 7: Workflow of the fault injection process

Algorithm 1 An algorithm for calculating the effect of faults in the input register

```
c_{out,f}^{start} as in Eq. (20) c_{out,f}^{end} as in Eq. (21) for co = c_{out,f}^{start} to c_{out,f}^{end} do
    y_0 \leftarrow 0 \text{ {main PE}}
    y_1 \leftarrow 0 \text{ {shadow PE}}
    for ci = 0 to C_{in} do
       for i=0 to H_k do
           for j=0 to W_k do
               if ci = w_{cf} and i = w_{if} and j = w_{if} then
                   y_0 \leftarrow y_0 + x_{(ci,u+i,v+j)} \cdot w_{(co,ci,i,j)} + e
                   y_1 \leftarrow y_1 + x_{(ci,u+i,v+j)} \cdot w_{(co,ci,i,j)}
               else
                   before the fault:
                   y_0 \leftarrow y_0 + x_{(ci,u+i,v+j)} \cdot w_{(co,ci,i,j)}
                   y_1 \leftarrow y_1 + x_{(ci,u+i,v+i)} \cdot w_{(co,ci,i,i)}
                   after the fault:
                   y_0 \leftarrow (y_0 \& y_1) + x_{(ci,u+i,v+i)} \cdot w_{(co,ci,i,i)}
                   y_1 \leftarrow y_1 + x_{(ci,u+i,v+j)} \cdot w_{(co,ci,i,j)}
               end if
           end for
       end for
    end for
    y_{(co,u,v)} = y_0
end for
```

VI. EVALUATION

A. Hardware parameters

The proposed architecture of the systolic array is described in SystemVerilog and synthesized using Cadence Genus 2021 and 45nm Nangate PDK. A comparison of the baseline systolic array without reconfiguration ability and four implementation options of FORTALESA is presented in Table IV. The size of the systolic arrays is 48×48. Input and weight registers are 8-bit long, and output register is 32-bit long.

TABLE IV: Hardware implementation parameters

Implementations	Area, mm ²	Power, W	Max frequency, MHz
Baseline SA	1.726	0.158	402
PM-DMR0-TMR3	1.937	0.177	357
PM-DMR0-TMR4	1.929	0.176	372
PM-DMRA-TMR3	2.129	0.193	303
PM-DMRA-TMR4	2.091	0.190	302

B. Reliability analysis

Two common CNN models were used for reliability evaluation, including AlexNet, trained on the CIFAR-10 dataset, and VGG-11, trained on the ILSVRC-2012 dataset. Models and datasets were selected to represent different network scales. For the fault injection experiments, networks were quantized to 8-bit integer format. Experiments were performed on NVIDIA GeForce RTX 3090 24G GPU.

Architectural vulnerability factor (AVF) was used as a reliability measure. AVF is the probability that a fault in the

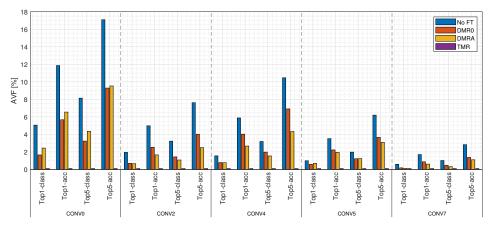


Fig. 8: Layer-wise reliability assessment of AlexNet considering transient faults

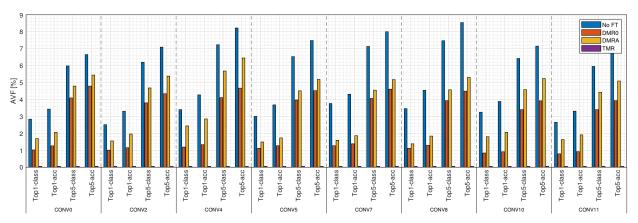


Fig. 9: Layer-wise reliability assessment of VGG-11 considering transient faults

hardware structure causes an application output error [41]. Since the output of the selected models is the probability score of each class, the following output errors of the DNNs were considered [23]:

- Top1-class: the top-ranked class is different from the golden run.
- Top1-acc: the probability score of the top-ranked class is different. The top-ranked class may be different. Top1-acc includes the Top1-class errors.
- Top5-class: at least one class in the top-5 is different, including different order of the top-5 classes.
- Top5-acc: the probability score of at least one class in the top-5 is different. The top-5 classes may be different. Top5-acc includes Top1-class, Top1-acc and Top5-class errors.

For transient fault analysis, layer-wise statistical fault injection was done following the equation introduced in [42] to achieve 95% confidence and 5% error margin. For each fault injection, fault parameters (Table II) were set randomly and a test dataset of 10.000 inputs was fed to the network.

Fig. 8 and 9 show AVF values of each layer for selected networks considering different execution modes and configurations of the 48×48 systolic array and transient faults. Faults were injected in each convolution layer using the proposed methodology. For TMR mode, it is assumed that all faults are corrected, and there are no output errors. Fully connected

layers were not considered for fault injection as they occupy only one row of the systolic array and their vulnerability factor, therefore, is very low [23].

Results show the varying vulnerability of different layers and the masking effect of DMR mode implementations. It can be seen that the DMR mode decreases the vulnerability of certain layers almost twice. Those results can be used to decide how to assign different layers of DNNs to different execution modes of the systolic array.

Fig. 10 shows AVF values for reliability assessment of permanent faults for AlexNet. The analysis is done for stuck-at-1 faults since they are proved to be more critical [23]. Unlike transient faults, analysis is done for the whole network (all convolutional layers) as permanent faults persist throughout the network execution.

C. Performance vs. reliability trade-off

A flexible performance vs. reliability trade-off can be achieved by executing different layers of the DNN using different modes of the reconfigurable systolic array. Fig. 11 and 12 plot reliability versus execution latency of the networks for all the possible combinations of modes. Latency is normalized to the execution of the network using performance mode for every layer (no fault tolerance). For reliability, Top1-class errors are considered. The Pareto front is marked with red color. Four implementations of FORTALESA are considered

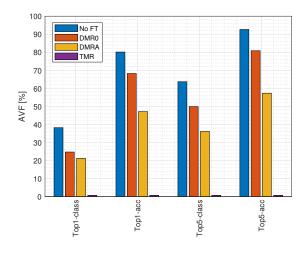


Fig. 10: Reliability assessment of AlexNet considering permanent faults (stuck-at-1)

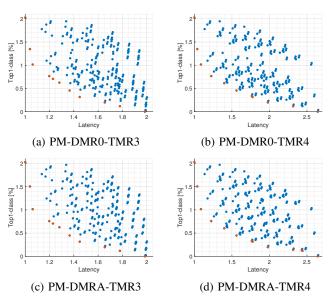


Fig. 11: Reliability vs. latency for AlexNet

for each network since different implementations of DMR mode give slightly different reliability results and different implementations of TMR mode affect the latency.

To compare different implementations of FORTALESA, the AVF of the Pareto front points was plotted against a product of latency, power, area, and delay (Fig. 13 and 14). It can be seen that while systolic arrays with the TMR4 implementation have smaller area and power, and higher frequency parameters, taking execution latency into consideration version with the TMR3 implementation shows better overall results.

D. Comparison with static redundancy

The comparison of the proposed run-time reconfigurable systolic array architecture with the static TMR approaches is presented in Fig. 15. A power-area product is plotted against the maximum possible throughput of the architecture (since the throughput of FORTALESA depends on the execution mode).

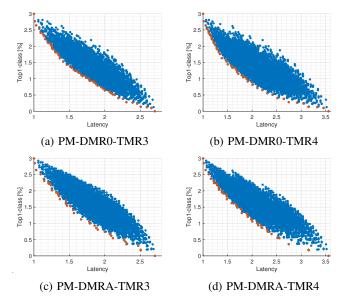


Fig. 12: Reliability vs. latency for VGG-11

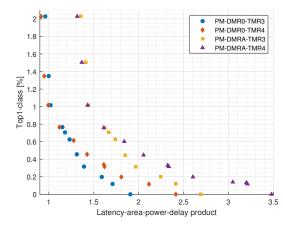


Fig. 13: Comparison of FORTALESA implementation options considering reliability for AlexNet

Throughput is calculated as the number of MAC operations performed by the systolic array in one clock cycle multiplied by the frequency. Different cases are considered for static TMR: triplication of registers only, triplication of registers and MAC units, and triplication of the whole array. As well as different sizes: 48×48 and 24×32 as this is the efficient size of the 48×48 systolic array operating in a TMR execution mode (TMR3 implementation option). As can be seen from the figure, the 24×32 systolic array with static TMR has a lower power-area product than the proposed architecture but has a fixed low throughput. On the other hand, the 48×48 systolic array with static TMR has high throughput, but the power-area product is significantly higher than the proposed architecture. FORTALESA, on average, requires $6 \times$ less resources than static TMR.

The proposed architecture is also compared with selective ECC proposed in [23]. While it protects only selected

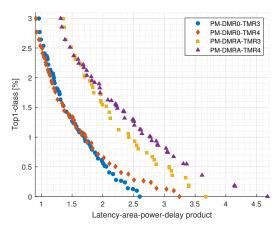


Fig. 14: Comparison of FORTALESA implementation options considering reliability for VGG-11

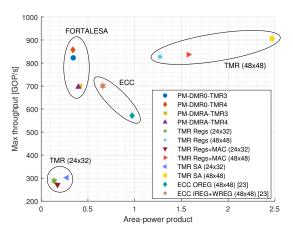


Fig. 15: Comparison of FORTALESA implementation options with static TMR and selective ECC [23]

registers, it requires, on average, $2.5\times$ more resources than FORTALESA, which protects all registers and MAC units.

E. Comparison with the state-of-the-art

The comparison of the proposed architecture with other methods to enhance fault tolerance of systolic arrays is presented in Table V. The proposed architecture protects both registers and MAC units in the PEs and since it utilizes a form of spatial redundancy it covers both transient and permanent faults. Faults are corrected in both fault-tolerant modes without interrupting inference execution for recovery.

VII. CONCLUSION

In this work, we proposed run-time reconfigurable fault-tolerant systolic array architecture FORTALESA with three execution modes and four implementation options. All four implementation options were evaluated in terms of resource utilization, throughput, and fault tolerance improvement. The proposed architecture is used for reliability enhancement of DNN inference on systolic array through heterogeneous mapping of different network layers to different execution modes.

TABLE V: Comparison with the state-of-the-art

Work	Covered faults		Protected parts		Fault
	Permanent	Transient	Registers	MAC	correction
[27]	√	Х	Х	1	√
[28]	✓	×	X	✓	\checkmark^1
[29]	✓	X	X	\checkmark^2	X
[23]	X	✓	✓	X	\checkmark^3
[30]	✓	×	✓	✓	X
[31]	X	\checkmark^4	X	✓	X
Our	✓	✓	✓	✓	✓

- ¹ Requires retraining of the DNN for each faulty chip.
- ² Faults in configurational memory of FPGA.
- ³ Only single-bit fault correction (ECC).
- ⁴ Timing faults from low voltage.

We also introduced a reliability assessment method based on fault propagation analysis for the evaluation of the proposed architecture and determination of the appropriate execution mode–layer mapping for DNN inference. The proposed architecture efficiently protects both registers and MAC units of systolic array PEs from transient and permanent faults without interrupting inference execution. The reconfigurability feature of FORTALESA enables a speedup up to $3\times$, depending on layer vulnerability, and requires $6\times$ less resources compared to static redundancy and $2.5\times$ less resources than the previously proposed solution for transient faults.

ACKNOWLEDGMENT

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS", CoE TK202 "Universum" and by EU Grant Project 101160182 "TAICHIP".

REFERENCES

- [1] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers et al., "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of* the 44th Annual International Symposium on Computer Architecture, 2017, p. 1–12.
- [2] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, 2016, pp. 262–263.
- [3] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao et al., "Gemmini: Enabling systematic deeplearning architecture evaluation via full-stack integration," in Proceedings of the 58th Annual Design Automation Conference (DAC), 2021.
- [4] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," ACM Comput. Surv., vol. 56, no. 6, 2024
- [5] X. Iturbe, B. Venu, E. Ozer, J.-L. Poupat, G. Gimenez, and H.-U. Zurek, "The arm triple core lock-step (TCLS) processor," ACM Transactions on Computer Systems, vol. 36, pp. 1–30, 2018.
- [6] M. Rogenmoser, N. Wistoff, P. Vogel, F. Gurkaynak, and L. Benini, "On-demand redundancy grouping: Selectable soft-error tolerance for a multicore cluster," in *IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, 2022, pp. 398–401.
- [7] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech, "Selective hardening for neural networks in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 66, pp. 216–222, 1 2019

- [8] C. Bolchini, L. Cassano, A. Miele, and A. Nazzari, "Selective hardening of CNNs based on layer vulnerability estimation," in *IEEE International* Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT, 2022.
- [9] T. G. Bertoa, G. Gambardella, N. J. Fraser, M. Blott, and J. McAllister, "Fault-tolerant neural network accelerators with selective TMR," *IEEE Design and Test*, vol. 40, pp. 67–74, 4 2023.
- [10] N. Khoshavi, A. Roohi, C. Broyles, S. Sargolzaei, Y. Bi, and D. Z. Pan, "SHIELDeNN: Online accelerated framework for fault-tolerant deep neural network architectures," in *Design Automation Conference*, 7 2020.
- [11] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "Enhancing fault resilience of QNNs by selective neuron splitting," in 2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS), 2023, pp. 1–5.
- [12] U. Zahid, G. Gambardella, N. J. Fraser, M. Blott, and K. Vissers, "FAT: Training neural networks for reliable inference under hardware faults," in *International Test Conference*, 11 2020.
- [13] J. Deng, Y. Fang, Z. Du, Y. Wang, H. Li, O. Temam, P. Ienne, D. Novo, X. Li, Y. Chen, and C. Wu, "Retraining-based timing error mitigation for hardware neural networks," in 2015 Design, Automation and Test in Europe Conference (DATE), 2015, pp. 593–596.
- [14] D. Xu, K. Xing, C. Liu, Y. Wang, Y. Dai, L. Cheng, H. Li, and L. Zhang, "Resilient neural network training for accelerators with computing errors," in 2019 IEEE 30th International Conference on Applicationspecific Systems, Architectures and Processors (ASAP), 2019, pp. 99– 102.
- [15] E. Ozen and A. Orailoglu, "Just say zero: Containing critical biterror propagation in deep neural networks with anomalous feature suppression," in *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*. Institute of Electrical and Electronics Engineers Inc., 11 2020.
- [16] E. Ozen and A. Orailoglu, "Boosting bit-error resilience of DNN accelerators through median feature selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, pp. 3250–3262, 11 2020.
- [17] L.-H. Hoang, M. A. Hanif, and M. Shafique, "FT-ClipAct: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *Design*, *Automation and Test in Europe*, 2020
- [18] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," in 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2021, pp. 1–13.
- [19] F. Geissler, S. Qutub, M. Paulitsch, and K. Pattabiraman, "A low-cost strategic monitoring approach for scalable and interpretable error detection in deep neural networks," in *Computer Safety, Reliability, and Security: 42nd International Conference, SAFECOMP*, 2023, p. 75–88.
- [20] B. F. Goldstein, V. C. Ferreira, S. Srinivasan, D. Das, A. S. Nery, S. Kundu, and F. M. Franca, "A lightweight error-resiliency mechanism for deep neural networks," in *International Symposium on Quality Electronic Design, ISQED*, 4 2021, pp. 311–316.
- [21] H. T. Kung, "Why systolic architectures?" Computer, vol. 15, no. 1, pp. 37–46, 1982.
- [22] E. Vacca, S. Azimi, and L. Sterpone, "A comprehensive analysis of transient errors on systolic arrays," in 2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2023, 2023, pp. 175–180.
- [23] J. Tan, Q. Wang, K. Yan, X. Wei, and X. Fu, "Saca-FI: A microarchitecture-level fault injection framework for reliability analysis of systolic array based CNN accelerator," *Future Generation Computer Systems*, vol. 147, pp. 251–264, 10 2023.
- [24] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim," in 2020 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2020, 8 2020, pp. 58–68.
- [25] U. K. Agarwal, A. Chan, A. Asgari, and K. Pattabiraman, "Towards reliability assessment of systolic arrays against stuck-at faults," in 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume, DSN-S 2023, 2023, pp. 230–236.
- [26] M. Taheri, M. Daneshtalab, J. Raik, M. Jenihhin, S. Pappalardo, P. Jimenez, B. Deveautour, and A. Bosio, "SAFFIRA: a framework for assessing the reliability of systolic-array-based DNN accelerators," in 2024 27th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2024, pp. 19–24.
- [27] Y. Zhao, K. Wang, and A. Louri, "FSA: An efficient fault-tolerant systolic array-based DNN accelerator architecture," in *IEEE International*

- Conference on Computer Design: VLSI in Computers and Processors, 2022, pp. 545–552.
- [28] J. J. Zhang, K. Basu, and S. Garg, "Fault-tolerant systolic array based accelerators for deep neural network execution," *IEEE Design and Test*, vol. 36, pp. 44–53, 10 2019.
- [29] F. Libano, P. Rech, and J. Brunhaver, "Efficient error detection for matrix multiplication with systolic arrays on FPGAs," *IEEE Transactions on Computers*, vol. 72, pp. 2390–2403, 8 2023.
- [30] E. Vacca, G. Ajmone, and L. Sterpone, "RunSAFER: A novel runtime fault detection approach for systolic array accelerators," in *The 41st IEEE International Conference on Computer Design*, 2023.
- [31] M. Safarpour, R. Inanlou, and O. Silvén, "Algorithm level error detection in low voltage systolic array," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, pp. 569–573, 2 2022.
- [32] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, S. Zheng, T. Lu, J. Gu, L. Liu, and S. Wei, "A high energy efficient reconfigurable hybrid neural network processor for deep learning applications," *IEEE Journal of Solid-State Circuits*, vol. 53, pp. 968–982, 4 2018.
- [33] C.-J. Lee and T. T. Yeh, "ReSA: reconfigurable systolic array for multiple tiny dnn tensors," ACM Transactions on Architecture and Code Optimization, vol. 37, p. 24, 9 2024.
- [34] C. Peltekis, D. Filippas, G. Dimitrakopoulos, C. Nicopoulos, and D. Pnevmatikatos, "ArrayFlex: A systolic array architecture with configurable transparent pipelining," in *Design, Automation and Test in Europe Conference (DATE 2023)*, 2023.
- [35] T. Zhao, S. Miao, S. Lu, J. Cao, J. Qiu, X. Shi, K. Wang, and L. He, "Towards a reconfigurable systolic array with multi-level packing for transformers," in *Architecture and System Support for Transformer Models (ASSYST@ ISCA 2023)*, 2023.
- [36] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016, pp. 367–379.
- [37] R. Xu, S. Ma, Y. Wang, and Y. Guo, "HeSA: heterogeneous systolic array architecture for compact CNNs hardware accelerators," in *Design*, *Automation and Test in Europe*, *DATE*, 2 2021, pp. 657–662.
- [38] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.
- [39] S. Burel, A. Evans, and L. Anghel, "MOZART+: Masking outputs with zeros for improved architectural robustness and testing of dnn accelerators," *IEEE Transactions on Device and Materials Reliability*, vol. 22, no. 2, pp. 120–128, 2022.
- [40] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in 10th International Workshop on Frontiers in Handwriting Recognition, 2006.
- [41] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, vol. 2003-January, pp. 29–40, 2003.
- [42] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *IEEE Design*, Automation and Test in Europe Conference and Exhibition (DATE), 2009, pp. 502– 506