# AdAM: Adaptive Approximate Multiplier for Fault Tolerance in DNN Accelerators

Mahdi Taheri[1*], Natalia Cherezova[1*], Samira Nazari[3], Ali Azarpeyvand[3,1], Tara Ghasempouri[1], Masoud Daneshtalab[1,2], Jaan Raik[1] and Maksim Jenihhin[1]

[1]Tallinn University of Technology, Tallinn, Estonia
[2]Mälardalen University, Västerås, Sweden
[3]University of Zanjan, Zanjan, Iran
[1]mahdi.taheri@taltech.ee

***Abstract*—Deep Neural Network (DNN) hardware accelerators are essential in a spectrum of safety-critical edge-AI applications with stringent reliability, energy efficiency, and latency requirements. Multiplication is the most resource-hungry operation in the neural network's processing elements. This paper proposes a scalable adaptive fault-tolerant approximate multiplier (AdAM) tailored for ASIC-based DNN accelerators at the algorithm and circuit levels. AdAM employs an adaptive adder that relies on an unconventional use of input Leading One Detector (LOD) values for fault detection by optimizing unutilized adder resources. A gate-level optimized LOD design and a hybrid adder design are also proposed as a part of the adaptive multiplier to improve the hardware performance. The proposed architecture uses a lightweight fault mitigation technique that sets the detected faulty bits to zero. The hardware resource utilization and the DNN accelerator's reliability metrics are used to compare the proposed solution against the Triple Modular Redundancy (TMR) in multiplication, unprotected exact multiplication, and unprotected approximate multiplication. It is demonstrated that the proposed architecture enables a multiplication with a reliability level close to the multipliers protected by TMR while at the same time utilizing 2.74× less area and with 39.06% less power-delay product compared to the exact multiplier. Moreover, it has similar area, delay, and power consumption parameters compared to the state-of-the-art approximate multipliers with similar accuracy while providing fault detection and mitigation capability.**

***Index Terms*—deep neural networks, approximate computing, circuit design, reliability, DNN accelerator**

## I. Introduction

In the past decades, Deep Neural Networks (DNNs) demonstrated a significant improvement in accuracy by adopting computationally intense models [1]. Consequently, the size of these models has increased drastically, imposing challenges in their deployment on resource-constrained platforms [2].

Different DNN compression techniques such as model quantization and pruning [2] as well as approximate computing (AxC) [3] [4] enable the use of DNNs in edge devices. While these techniques decrease the accuracy of DNNs, they bring the benefits of lower resource utilization and energy consumption and higher system efficiency [5]. As an example, quantizing DNNs down to 8-bit INT gained popularity in edge AI applications, because of the minimal impact on accuracy drop and a significant reduction in memory footprint [6].

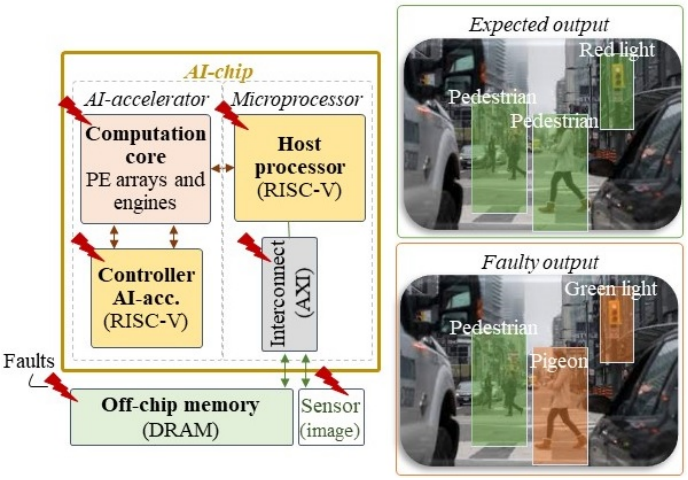


Fig. 1: Possible locations of faults in AI chips [7]

On the other hand, the role of DNNs in a wide range of safety- and mission-critical applications e.g., autonomous driving, is expanding. Therefore, deploying a DNN accelerator requires addressing the trade-off between different design parameters and *reliability* [4]. Although DNNs possess certain intrinsic fault-tolerant and error-resilient characteristics, it is insufficient to conclude the reliability of DNNs without considering the characteristics of the corresponding hardware accelerator.

Consider Fig. 1 that demonstrates different possible fault locations in an AI accelerator and their negative effect on the object detection task. In the example, a pedestrian has been identified as a bird, and a red light is misclassified as a green one leading to a potentially disastrous situation. In this work, which is an extension of the previous paper by the authors [8], the faults in the computational core of the AI chip are considered.

Moreover, with the continuous scaling-down of the process, there is a discernible trend indicating that the Soft Error Rate (SER) of combinational circuits may surpass that of sequential

* These authors contributed equally

circuits [9], [10]. Therefore, the main focus of this study is introducing a novel reliability technique to mitigate soft errors in the combinational logic of DNN accelerators while keeping resource utilization and energy consumption low.

This work presents an architecture of an adaptive fault-tolerant approximate multiplier (AdAM) tailored for ASIC-based DNN accelerators. The multiplier is based on the logarithmic Mitchell's multiplier that substitutes multiplication with the addition of approximated logarithms of the operands. The proposed multiplier protects higher-order bits of the product based on the maximum position of the leading one bit in the input strands of the multiplier. This multiplier is a negative overhead fault tolerance approximate multiplier compared to the exact multipliers. The contributions of the paper are as follows:

- The architecture of a novel adaptive fault-tolerant approximate multiplier tailored for DNN accelerators, including an adaptive adder relying on an unconventional use of the leading one position value of the inputs for fault detection through optimizing unutilized adder resources.
- Implementation and validation of the multiplier design.
- Reliability and hardware performance trade-off assessment and comparison of the proposed multiplier with exact and approximate state-of-the-art multipliers using several state-of-the-art DNN benchmarks.

The proposed multiplier provides a reliability level close to the multipliers protected by Triple Modular Redundancy (TMR) while utilizing 2.74× less area and having 39% less power-delay product compared to the unprotected exact multiplier. In fact, it has similar area, delay, and power consumption parameters compared to the state-of-the-art approximate multipliers with similar accuracy while providing fault detection and mitigation capability.

The remainder of the paper is organized as follows: Section II summarizes related works. Section III presents the proposed method. Section IV presents the experimental setup and discusses the results. Finally, Section V concludes the work.

## II. Related Work

Multipliers are one of the primary arithmetic building blocks widely used in DNNs. Approximate computing is a promising technique for designing digital circuits with lower area and power consumption while achieving a higher working frequency, particularly when the target application has some error resiliency, such as DNNs [3], [5]. Various approximate multipliers are proposed in the literature. These options encompass dynamic segment multiplier (DSM) structure [11], dynamic range unbiased multiplier (DRUM) structure [12], memristor-based multipliers [3], [13], simplified adder-based or truncated multipliers [14], utilization of inexact compressors [15], and approximate booth multipliers [16]. One of the other approximation techniques to further speed up the multiplication is to move to the logarithmic numbering system to compute addition instead of multiplication. The general idea is to obtain the logarithm of the inputs, calculate their sum, and convert the output value to the final result through an antilogarithm operation [17], [18]. The complexity and accuracy of this method come from the logarithm and antilogarithm steps. Truncating the input operand of multiplication and using logarithmic approximation also has a dual effect on the area, speed, and power consumption improvement. The first approximate logarithmic multiplier was proposed by Mitchell, who used binary logarithms to approximate multiplication [18]. Several studies have been conducted on improving the performance and accuracy of this method by considering DNN application [19]. TOSAM is a scalable approximate multiplier that reduces the number of partial products by truncating each of the input operands based on their leading one-bit position and improves delay, area, and energy consumption up to 41%, 90%, and 98% respectively [14]. ScaleTRIM is a scalable approximate unsigned LOD multiplier for DNNs that exploits curve fitting and linearization for fitting input products and a novel error compensation method using lookup tables [17]. More details about recent works on the approximate arithmetic circuits can be found in [20] and specifically about approximation for DNNs in [21].

The error introduced by the approximation is deterministic, and its impact can be studied comprehensively on the accuracy drop of the network. However, soft errors are unpredictable in contaminated and harsh environments that can lead to DNNs malfunction and accuracy drop drastically [22]. The design of approximate arithmetic circuits and their reliability have been extensively covered in the literature both without considering an application [23], [24], [25] and concerning DNNs [4], [5], [26], [27], [28], [29]. For example, the work presented in [30] introduces a comprehensive library of approximate adders and multipliers, primarily intended for use in image and video processing circuits. However, as highlighted in [31], there is a trade-off between fault resilience and energy efficiency when using the multipliers from this library. This indicates the need for specific approximate multiplier (AxM) designs to optimize energy savings and resource utilization while enhancing DNN accuracy. Additionally, the impact of approximate multipliers on DNN accuracy can vary significantly, with some leading to a substantial drop in performance. In contrast to the proposed approximate multipliers, AdAM considers reallocating resources saved by approximation for fault tolerance.

Traditional fault tolerance methods for DNN hardware accelerators include hardware redundancy, correction codes for parameter memories, and algorithm-based fault tolerance. Triple Modular Redundancy (TMR), despite achieving 100% fault coverage, incurs a substantial near 200% area and power overhead [32]. Therefore, numerous algorithms and frameworks are developed to enhance the efficiency of applying these methods and balance their hardening effects and design costs such as Approximate TMR (ATMR) [33]. ATMR is a technique that replaces some modules of TMR with approximate ones while ensuring that the majority voter gives the correct output [34]. However, ATMR still requires duplicating the whole combinational circuit, even at the finest level of granularity [35], [36].

To address the high overhead incurred by traditional fault

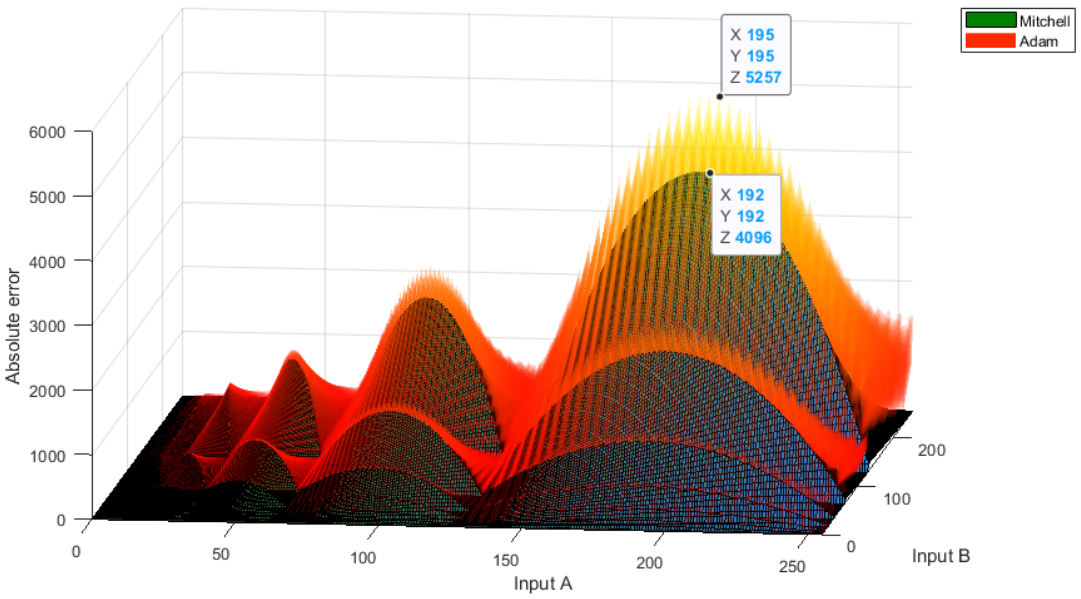


Fig. 2: Comparison of absolute error distribution between the original 8-bit Mitchell's multiplier and AdAM: top point shows the maximum absolute error for Adam, lower point shows the maximum absolute error for Mitchell's multiplier

tolerance methods, this work presents an adaptive approximate multiplier that provides a high level of reliability while using less area and PDP than an exact multiplier.

## III. AdAM architecture

We propose an architecture for adaptive fault-tolerant approximate multiplier tailored for DNN accelerators. This architecture includes an adaptive adder relying on an unconventional use of input Leading One Detector (LOD) values for fault *detection* and *mitigation* through the optimization of unutilized adder resources. A gate-level optimized LOD design and a lightweight triplicated hybrid adder design are used to enhance further the proposed architecture's reliability, resource utilization, and efficiency. The base for the proposed multiplier is the classical Mitchell multiplier [18]. However, the methodology can be applied to all logarithmic approximate multipliers. Another level of approximation is introduced in the adaptive adder (Fig. 3) considering the application of this multiplier in DNNs with a proven negligible impact on the network accuracy (see Table VI in the results section). Mitchell multiplier employs approximate logarithms of the input values. By adding these logarithms, Mitchell's algorithm estimates the product. The final result is obtained by taking the antilogarithm of this sum.

### A. Definitions

Consider two $n$-bit positive integers $A$ and $B$ that can be represented as

$$A = \sum_{i=0}^{k_A} 2^i a_i, B = \sum_{i=0}^{k_B} 2^i b_i, \quad a_i = b_i = \{0, 1\} \tag{1}$$

where $k$ is the position of the leading one bit, $0 \leq k < n$. By factoring $2^k$ we get

$$A = 2^{k_A}\left(1 + \sum_{i=0}^{k_A-1} 2^{i-k_A} a_i\right) = 2^{k_A}(1 + X_A) \tag{2}$$

$$B = 2^{k_B}\left(1 + \sum_{i=0}^{k_B-1} 2^{i-k_B} b_i\right) = 2^{k_B}(1 + X_B) \tag{3}$$

Since $k \geq 0$, $X$ is in the range $0 \leq X < 1$ and is the fraction term of the number. The logarithms of $A$ and $B$ then can be expressed as

$$log(A) = k_A + log(1 + X_A) \tag{4}$$

$$log(B) = k_B + log(1 + X_B) \tag{5}$$

Mitchell's method approximates $log(1+X)$ with the value of $X$. This way, logarithms are approximated as

$$log(A) \approx k_A + X_A \tag{6}$$

$$log(B) \approx k_B + X_B \tag{7}$$

The logarithm of $A * B$ can be approximated as

$$log(A * B) \approx k_A + X_A + k_B + X_B \tag{8}$$

The antilogarithm of the above expression is the final product:

$$\hat{P} = 2^{k_A + k_B}(1 + X_A + X_B) \tag{9}$$

AdAM architecture introduces a truncation parameter $t$ that defines the level of fault tolerance, i.e. the number of pro-

tected higher-order bits of the fractional part. Considering the truncation parameter, fractional part $X$ is defined as

$$X_A^t = \sum_{i=0}^{k_A-1} 2^{i-k_A} a_i \cdot [|i-k_A| \geq n-t] \quad (10)$$

$$X_B^t = \sum_{i=0}^{k_B-1} 2^{i-k_B} b_i \cdot [|i-k_B| \geq n-t] \quad (11)$$

where $n$ is the number of bits, $t$ is the truncation parameter, $[\cdot]$ are the Iverson brackets. The final product then is expressed as

$$\hat{P} = 2^{k_A+k_B}(1 + X_A^t + X_B^t) \quad (12)$$

Truncation of the fractional parts introduces additional errors to the result. The absolute error $(P - \hat{P})$ behavior of the original 8-bit Mitchell's multiplier and AdAM is compared in Fig. 2 over the entire input domain. The maximum errors introduced by each method are marked in the figure. There is no additional error when the truncated bits in both operands are '0', as the number of '1' in the truncated bits increases, additional error also increases.

However, it has been shown in the literature that the trained synaptic weights in NN applications do not have a uniform distribution, and they are mostly centered around zero [37]. Since the proposed multiplier presents a higher level of protection for smaller values respecting the value of LOD, it can effectively protect most computations of the network during inference.

The impact of the proposed multiplier on the accuracy of different networks is reported in the result section.

The level of fault tolerance of the proposed multiplier is defined by two parameters: the aforementioned truncation parameter $t$ and duplication level $h$. The truncation parameter $t$ defines the minimum number of protected bits, and duplication level $h$ defines the maximum possible number of protected bits. Smaller operands have smaller mantissa values that do not require the whole adder. Therefore, more adder resources can be used for fault tolerance by duplicating the addition of more higher-order bits. Duplication level $h$ does not affect the accuracy of the multiplier, as a hardware-related parameter it affects area, power, and delay. Duplication level $h$ defines the amount of adder resources that can be used for fault tolerance (see Subsection III-C for detailed description). A higher $h$ value means a higher level of fault tolerance and higher resource utilization. A smaller value means a lower level of fault tolerance and lower resource utilization. Different configurations of the proposed multiplier are denoted as AdAM($t$,$h$).

### B. Hardware implementation

The proposed architecture of the multiplier is presented in Fig. 3 (the contributions and extensions to the logarithmic Mitchell multiplier are marked with red color). First, a novel optimized *Leading One Detector (LOD)* circuit (subsection III-D) is used to find the index of the first '1' bit in each operand. This index denoted as $k$, is the characteristic or

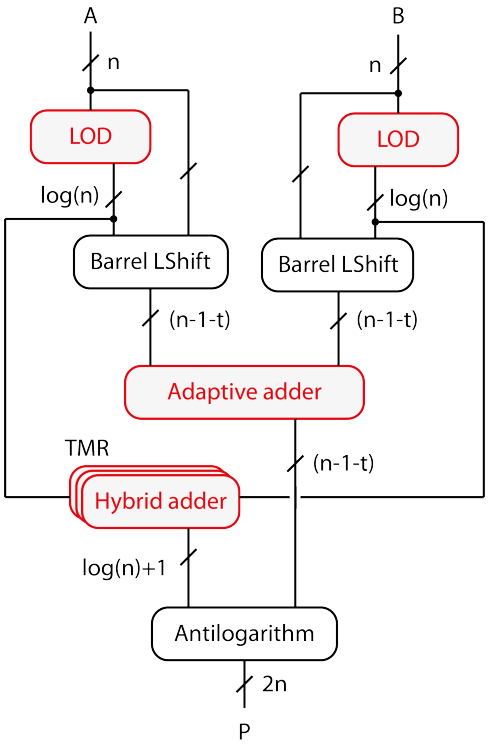


Fig. 3: AdAM architecture (the contributions and extensions to the logarithmic Mitchell multiplier are marked with red color)

integer part of the logarithm and has $log_2(n)$ bits. Then, the operands are shifted left by $k$ bits, aligning the leading one with the Most Significant Bit (MSB). $(n–1)$ bits after the leading one represent the mantissa part denoted as $m$. The mantissa is truncated to $(n-1-t)$ bits. The truncated operands are passed to the adaptive $(n–1)$-bit adder that adds mantissa together and duplicates the addition of several higher-order bits depending on the $k$ value of the biggest operand for fault *detection* and *mitigation*. As it was already mentioned, the number of duplicated bits depends on truncation parameter $t$ and duplication level $h$.

The architecture of the *adaptive adder* is shown in Fig. 4 (subsection III-C). The adder is based on the carry-lookahead adder, and the carry generation logic is excluded from the figure to save space. Duplicated results are compared, and if there is a fault, the faulty bit is set to zero using AND gates (marked on the figure with a red rectangle). Due to the truncation of the mantissa, up to $t$ lower-order bits are excluded from the calculation, which affects only the bigger numbers with the $k > (n-t)$. This introduces a small error compared to the original Mitchell algorithm (discussed in the following section).

The $k$ values of the operands are added separately using a small *hybrid adder*. A hybrid between a resource-intensive but fast carry look-ahead adder and a lightweight but slow due to the accumulated delay ripple carry (CR) adder is selected for this task. This adder is replicated three times, for extra fault tolerance, as the order of the final output depends on the

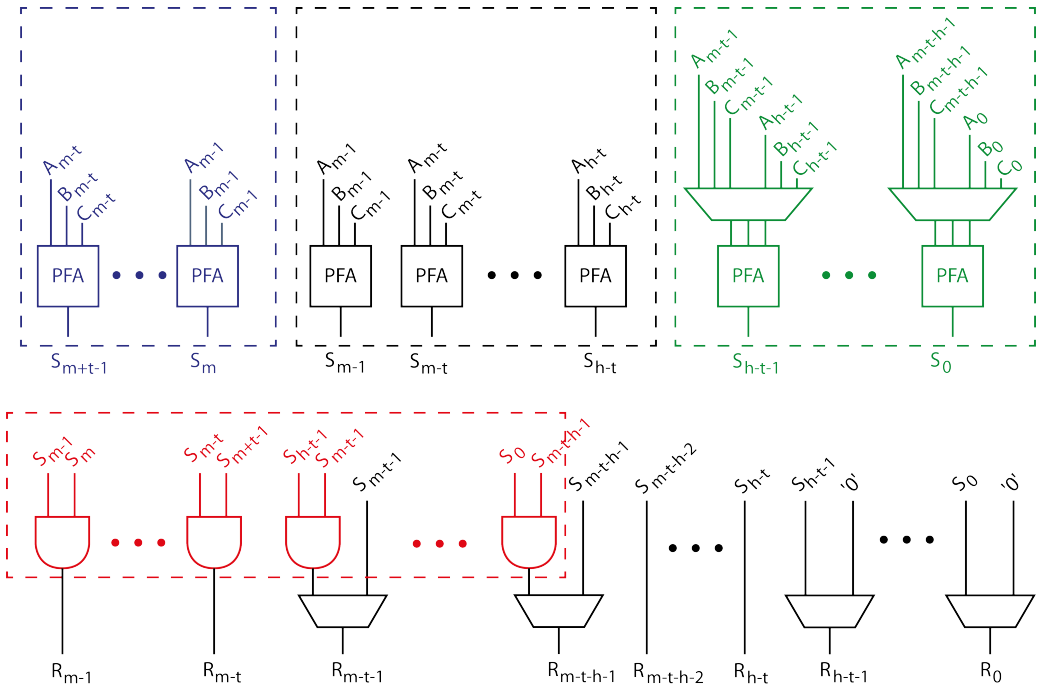


Fig. 4: Adaptive adder architecture: $A$ and $B$ are inputs, $C$ is carry values, and PFA stands for partial full adder, $m$ here denotes the size of the truncated mantissa $(n-1-t)$. The truncation parameter $t$ defines the minimum number of protected bits, $t$ PFAs that duplicate the addition of $t$ higher-order bits of mantissa are colored blue. The duplication level $h$ defines the maximum number of protected bits, $h-t$ PFAs with multiplexers that can duplicate the addition of higher-order bits or add lower-order bits are colored green.

result of this addition. A majority voter selects the final result. Then, the antilogarithm algorithm is used to get the product of multiplication. The sum of $k$ values determines the position of the leading one in the output product, followed by the sum of the mantissa parts using the appropriate shift operation.

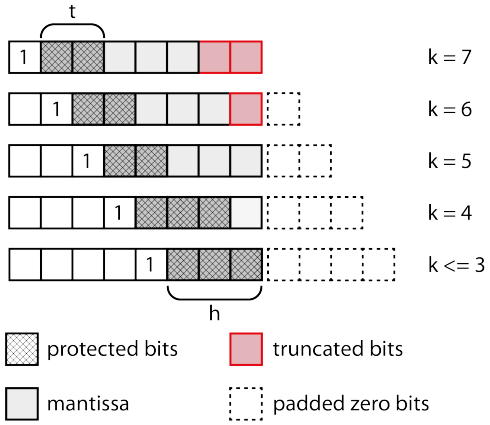


Fig. 5: Fault-tolerance and error introduced based on different input values $(n=8, t=2, h=3)$

### C. Adaptive adder

The adaptive adder is designed to detect and mitigate faults based on the multiplier inputs' $k$ values. The adder is divided into three parts: $t$ PFAs (Partial Full Adders) for duplicating the addition of $t$ higher-order bits (colored blue in Fig. 4), $(h-t)$ PFAS with multiplexers on inputs that can duplicate the addition of up to $(h-t)$ higher-order bits or add lower-order bits (colored green in Fig. 4), and $(n-1-h)$ PFAs for the addition of the mantissa. This way, for smaller operands that have smaller mantissa and do not require the whole adder, more resources can be used for fault tolerance. The size of the truncated mantissa can be expressed as $W_{X^t} = k \cdot [k < n-t] + (n-1-t) \cdot [k \geq n-t]$ using Iverson brackets notation. The available adaptive adder resources can be expressed as $W_{AR} = (n-1) - W_{X^t}$. Thus, the number of protected bits is $W_{PB} = W_{AR} \cdot [W_{AR} < h] + h \cdot [W_{AR} \geq h]$.

As an example, Fig. 5 shows the scheme in which the proposed multiplier introduces fault tolerance considering 8-bit inputs with a truncation parameter $t = 2$ and duplication level $h = 3$. As shown in this figure, five cases are considered. If the maximum LOD of the inputs is 7, two lower-order bits are discarded, and a two-bit adder of the adaptive adder is dedicated to recomputing the addition of two higher-order bits. These results are compared, and the mismatched bits are set to zero.

For $k = 6$, only the Least Significant Bit (LSB) is discarded, and two higher-order bits are protected the same way as in the previous case. When $k = 5$, no bits are discarded, and two higher-order bits are protected. For $k = 4$, three higher-order bits are protected, and only LSB is not monitored. In the case of $k \leq 3$, all bits are protected, enabling the

proposed multiplier to provide comprehensive fault detection and mitigation for all inputs.

### D. LOD design

The gate-level structure of the proposed LOD circuit is presented in Fig. 6. The circuit is divided into two functional parts: zero flag calculation and leading-one position detection (LOPD). LOPD consists of several stages. During the first stage, input is divided into nibbles that are processed in parallel. For each nibble, three signals are calculated except for the first one: $a$, $b$, and $c$. Only two signals are calculated for the first nibble: $b$ and $c$. Signal $a$ defines whether there is a '1' bit in the nibble; signal $b$ defines whether the output is even or odd; and signal $c$ defines whether there is '1' in the two higher bits of the nibble. During the second stage, those signals form the final result based on the highest nibble with a '1' bit. Since the zero flag is only required at the last stage of the multiplier to determine whether the final product should be zero, it is calculated using the LOPD output values. The proposed design requires fewer logic gates (as demonstrated in the results section) than designs found in the literature (e.g., ScaleTRIM [17]) by reusing the output values for calculating the zero flag, knowing that it can have a longer delay.

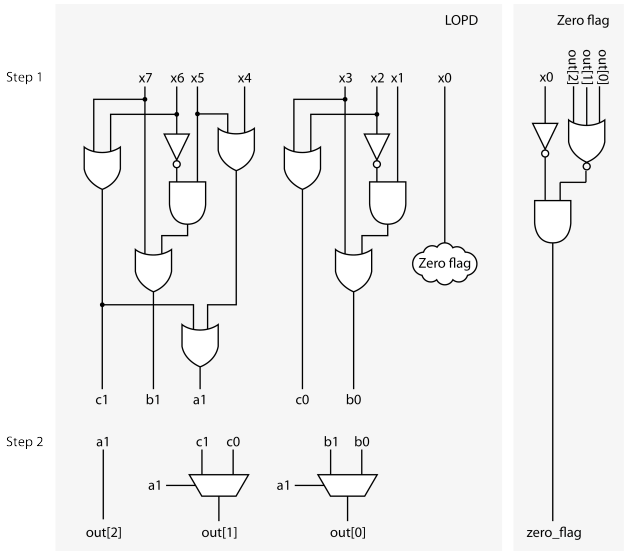


Fig. 6: Gate-level structure of the proposed LOD

## IV. Experimental Results

### A. Experimental Setup

In this paper, the FreePDK 45 nm Nangate technology library is used in Cadence Genus 2023 to compare the hardware characteristics (area, latency, power consumption) of the proposed methods with the state-of-the-art. The accuracy is reported using Mean Absolute Relative Error (MARE) calculated as

$$\text{MARE} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{P_i - \hat{P}_i}{P_i}\right|,$$

where $N$ is the number of tested input combinations, $P_i$ and $\hat{P}_i$ are the exact and approximate results.

Additionally, a design of a MAC (multiply-accumulate) unit in a systolic array is synthesized for ASIC to better illustrate the results in an AI core. The impact on the accuracy of the proposed adaptive multiplier is studied on different networks (i.e., LeNet-5, AlexNet, ResNet-18, VGG-16, DenseNet) trained on MNIST and CIFAR-10 using 8-bit INT with the help of the AdaPT framework [38]. Finally, the impact of the proposed multiplier on the reliability of DNNs is studied using the mentioned benchmarks.

*Fault injection.* Fault injection is performed, assuming the single bit-flip faults in the network's MAC operation of a systolic array for reliability assessment. According to the adopted single-bit fault model, a random bit-flip is injected into a random MAC unit of the systolic array core at a random execution time of the network, and the whole test set is fed to the network to obtain the accuracy of the network. Transient faults (bit flips) within subcomponents of the multipliers, such as LOD, adders, and barrel shifters, were considered for the fault injection experiments. An analysis was done to determine how those faults will propagate to the output of the multiplier. This way, for each considered fault, a look-up table was generated with the output for every input combination. Those look-up tables were used during the fault injection. The number of faults injected was chosen based on preliminary experiments and existing literature. Using equations provided in [39] a 95% confidence level and 1% error margin were reached.

### B. Hardware utilization

In this section, the proposed LOD, and adaptive multiplier are compared in terms of power and area with state-of-the-art designs. Power–Delay Product (PDP) is used to show the efficiency of the design.

Table I shows the area, power, and delay of the proposed 8-bit LOD architecture compared to the state-of-the-art design. The proposed design has fewer gates and a smaller critical path compared to the other methods in the literature.

TABLE I: LOD hardware comparison between the proposed method and the state-of-the-art

| LOD Architecture | Delay (ps) | Power ($\mu$W) | PDP (ps$\times\mu$W) | Area ($\mu m^2$) |
|---|---|---|---|---|
| ScaleTrim [17] | 156 | 1.12 | 174.72 | 9.84 |
| Proposed | 136 | 1.09 | 148.24 | 9.57 |

Tables II, III and IV report the accuracy, efficiency, and fault tolerance (FT) of 8-bit, 16-bit, and 32-bit approximate multipliers compared with the proposed method. Wallace, DRUM [12], TOSAM [14], and ScaleTrim [17] are used for this comparison. Logarithmic AxMs such as TOSAM show better performance compared to other state-of-the-art non-logarithmic AxMs, including DSM [11], ROBA [40], and LETAM [41]. For instance, TOSAM structures exhibit significantly lower energy consumption and area utilization compared to DRUM,

TABLE II: Accuracy and efficiency of 8-bit approximate multipliers compared with the proposed method

| Multiplier Architecture | Delay (ns) | Power ($\mu$W) | Area ($\mu m^2$) | MARE (%) | FT | PDP (pJ) |
|---|---|---|---|---|---|---|
| Exact (Wallace) | 0.85 | 360 | 417 | 0.00 | No | 306 |
| DRUM(3) | 0.70 | 104 | 143 | 12.6 | No | 72.8 |
| TOSAM(0,2) | 0.58 | 120 | 186 | 10.1 | No | 69.6 |
| TOSAM(0,3) | 0.68 | 144 | 198 | 7.7 | No | 97.9 |
| DRUM(4) | 1.00 | 172 | 208 | 6.4 | No | 172 |
| TOSAM(1,5) | 0.88 | 231 | 291 | 4.1 | No | 203.2 |
| AdAM(2,3) | 1.13 | 165 | 152 | 4.7 | Yes | 186.4 |
| ScaleTrim(4,8) | 1.8 | 143 | 216 | 3.3 | No | 257.4 |

TABLE III: Accuracy and efficiency of 16-bit approximate multipliers compared with the proposed method

| Multiplier Architecture | Delay (ns) | Power (mW) | Area ($\mu m^2$) | MARE (%) | FT | PDP (pJ) |
|---|---|---|---|---|---|---|
| Exact (Wallace) | 1.22 | 2.08 | 1785 | 0.00 | No | 2537.6 |
| DRUM(3) | 0.88 | 0.13 | 257 | 11.9 | No | 114.4 |
| TOSAM(0,2) | 0.74 | 0.16 | 342 | 10.9 | No | 118.4 |
| ScaleTrim(3,4) | 1.35 | 0.20 | 281 | 9.23 | No | 279.0 |
| TOSAM(0,3) | 0.84 | 0.21 | 423 | 7.6 | No | 176.4 |
| DRUM(4) | 1.12 | 0.27 | 381 | 5.9 | No | 302.4 |
| ScaleTrim(7,0) | 2.38 | 0.36 | 492 | 4.06 | No | 871.3 |
| TOSAM(1,5) | 1.00 | 0.35 | 532 | 4.0 | No | 350.0 |
| AdAM(6,7) | 1.00 | 0.10 | 440 | 3.97 | Yes | 100.0 |
| AdAM(4,7) | 1.06 | 0.13 | 451 | 3.87 | Yes | 137.8 |
| AdAM(4,4) | 0.96 | 0.12 | 434 | 3.87 | Yes | 115.2 |
| AdAM(2,7) | 1.32 | 0.15 | 495 | 3.97 | Yes | 171.6 |
| AdAM(2,4) | 1.13 | 0.13 | 451 | 3.85 | Yes | 146.9 |
| DRUM(5) | 1.36 | 0.43 | 532 | 2.9 | No | 584.8 |
| ScaleTrim(9,0) | 2.71 | 0.43 | 541 | 2.2 | No | 1170.6 |
| TOSAM(2,6) | 1.21 | 0.38 | 564 | 2.1 | No | 459.8 |

DQ4:2C4, and AS-RoBA multipliers, while maintaining comparable accuracy. Specifically, the TOSAM(1,5) configuration achieves 4% lower delay, 78% lower area, and 70% lower energy consumption compared to the RoBA multiplier. The results also show that the overall impact of the TOSAM multiplier, one of the baseline approaches in this paper, on delay, power, area, and MARE, are lower than those of other approximate multipliers with similar MARE values. Therefore, to respect the page limits of the paper, we did not include other approximate multipliers in our comparison.

The proposed multiplier has similar hardware parameters to the state-of-the-art approximate multipliers with similar accuracy while providing reliability improvement with fault detection and mitigation capability.

Table V shows the results of MAC implementation with different multipliers. The MAC unit with the proposed multiplier takes less area and has a lower PDP value.

### C. DNN accuracy

Table VI compares the accuracy of different CNN architectures using the proposed approximate multiplier with the baseline accuracy using the exact multiplier. The results in Table VI were obtained by directly substituting the accurate multiplier with the proposed multiplier without any retraining or fine-tuning. The evaluation shows that the accuracy of DNN with the proposed method is very close to the baseline. Hence,

TABLE IV: Accuracy and efficiency of 32-bit approximate multipliers compared with the proposed method

| Multiplier Architecture | Delay (ns) | Power (mW) | Area ($\mu m^2$) | MARE (%) | FT | PDP (pJ) |
|---|---|---|---|---|---|---|
| Exact (Wallace) | 1.67 | 10.26 | 7618 | 0.00 | No | 17134.2 |
| DRUM(3) | 1.08 | 0.20 | 520 | 11.90 | No | 216.0 |
| TOSAM(0,2) | 0.99 | 0.27 | 844 | 10.90 | No | 267.3 |
| TOSAM(0,3) | 1.02 | 0.28 | 780 | 7.61 | No | 285.6 |
| DRUM(4) | 1.33 | 0.36 | 738 | 5.90 | No | 478.8 |
| TOSAM(1,5) | 1.18 | 0.40 | 999 | 3.95 | No | 472.0 |
| AdAM(8,15) | 1.67 | 0.31 | 1105 | 3.8488 | Yes | 517.7 |
| AdAM(8,8) | 1.30 | 0.29 | 1041 | 3.8488 | Yes | 377.0 |
| AdAM(6,15) | 1.72 | 0.48 | 1148 | 3.8487 | Yes | 825.6 |
| AdAM(6,8) | 1.46 | 0.33 | 1141 | 3.8487 | Yes | 471.9 |
| AdAM(4,15) | 2.43 | 0.57 | 1425 | 3.8487 | Yes | 1385.1 |
| AdAM(4,8) | 1.80 | 0.46 | 1403 | 3.8487 | Yes | 828.0 |
| AdAM(2,15) | 2.49 | 0.59 | 1579 | 3.8487 | Yes | 1469.1 |
| AdAM(2,8) | 2.02 | 0.49 | 1519 | 3.8487 | Yes | 989.8 |
| DRUM(5) | 1.55 | 0.56 | 944 | 2.89 | No | 868.0 |
| TOSAM(2,6) | 1.30 | 0.54 | 1146 | 2.06 | No | 702.0 |
| DRUM(6) | 1.69 | 0.75 | 1059 | 1.47 | No | 1267.5 |
| TOSAM(3,7) | 1.44 | 0.69 | 1294 | 1.05 | No | 993.6 |
| DRUM(7) | 1.85 | 0.96 | 1235 | 0.73 | No | 1776.0 |
| TOSAM(4,8) | 1.57 | 0.83 | 1411 | 0.53 | No | 1303.1 |
| DRUM(8) | 1.93 | 1.17 | 1402 | 0.37 | No | 3545.2 |
| TOSAM(5,9) | 1.60 | 1.08 | 1625 | 0.27 | No | 1728.0 |

TABLE V: Efficiency of 8-bit MAC unit of a systolic array with different multipliers

| MAC Architecture | Delay (ns) | Power (mW) | PDP (ns$\times$mW) | Area ($\mu m^2$) |
|---|---|---|---|---|
| Exact (Wallace) | 1151 | 2.10 | 2417.10 | 975.95 |
| ScaleTrim(4,8) | 1106 | 2.00 | 2212.00 | 949.35 |
| AdAM(2,3) | 1098 | 1.47 | 1614.06 | 794.80 |

TABLE VI: Accuracy comparison of different CNNs with an exact (baseline) and the proposed approximate multiplier

| DNN | Accuracy with Wallace (%) | Accuracy with AdAM(2,3) (%) |
|---|---|---|
| LeNet-5 (MNIST) | 93.8 | 94.1 |
| AlexNet (MNIST) | 78.0 | 77.7 |
| VGG-11 (CIFAR-10) | 93.4 | 94.0 |
| VGG-13 (CIFAR-10) | 92.0 | 92.0 |
| VGG-19 (CIFAR-10) | 92.0 | 93.0 |
| ResNet-18 (CIFAR-10) | 93.8 | 93.2 |
| ResNet-34 (CIFAR-10) | 93.0 | 94.0 |
| ResNet-50 (CIFAR-10) | 95.0 | 94.0 |
| DenseNet (CIFAR-10) | 92.6 | 95.0 |
| Inception (CIFAR-10) | 92.6 | 95.0 |

the proposed multiplier has a negligible effect on the accuracy of DNNs. It is observed that the accuracy of approximated networks can sometimes exceed that of exact networks. The observed higher accuracy with AdAM compared to the exact multiplier in some cases can be attributed to the probabilistic nature and inherent non-linearity of DNNs, as well as the sensitivity of certain bit positions to errors. This observation aligns with the previous studies [42] showing that errors in the arithmetics logic can result in better inference accuracy and consequently referring to them as "good faults". This behavior arises because faults, particularly those affecting

TABLE VII: Fault coverage (SDC-5%) in different benchmarks

| SDC-5% | AlexNet | DenseNet | Inception | VGG-11 | VGG-13 | VGG-19 | ResNet-18 | ResNet-34 | ResNet-50 |
|---|---|---|---|---|---|---|---|---|---|
| **Unp-Exact** | 70.9 | 17.2 | 21.8 | 9.2 | 10.8 | 10.8 | 10.8 | 17.2 | 16.2 |
| **Unp-AxM** | 63.5 | 15.4 | 20.6 | 6 | 7.2 | 17.8 | 9.4 | 13.6 | 12.2 |
| **Pro-TMR** | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| **Pro-AdAM** | 99.6 | 42.2 | 41.8 | 36.6 | 32.6 | 57.2 | 37.6 | 44.8 | 38.2 |

TABLE VIII: Fault coverage (SDC-20%) in different benchmarks

| SDC-20% | AlexNet | DenseNet | Inception | VGG-11 | VGG-13 | VGG-19 | ResNet-18 | ResNet-34 | ResNet-50 |
|---|---|---|---|---|---|---|---|---|---|
| **Unp-Exact** | 90.3 | 91.6 | 76.6 | 77.4 | 81 | 81 | 84 | 86.8 | 86 |
| **Unp-AxM** | 78.5 | 92.2 | 75.8 | 76 | 78.8 | 75 | 80 | 84.8 | 82.4 |
| **Pro-TMR** | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| **Pro-AdAM** | 99.8 | 96.6 | 89.2 | 94.8 | 88.8 | 86.4 | 90.6 | 91 | 92.6 |

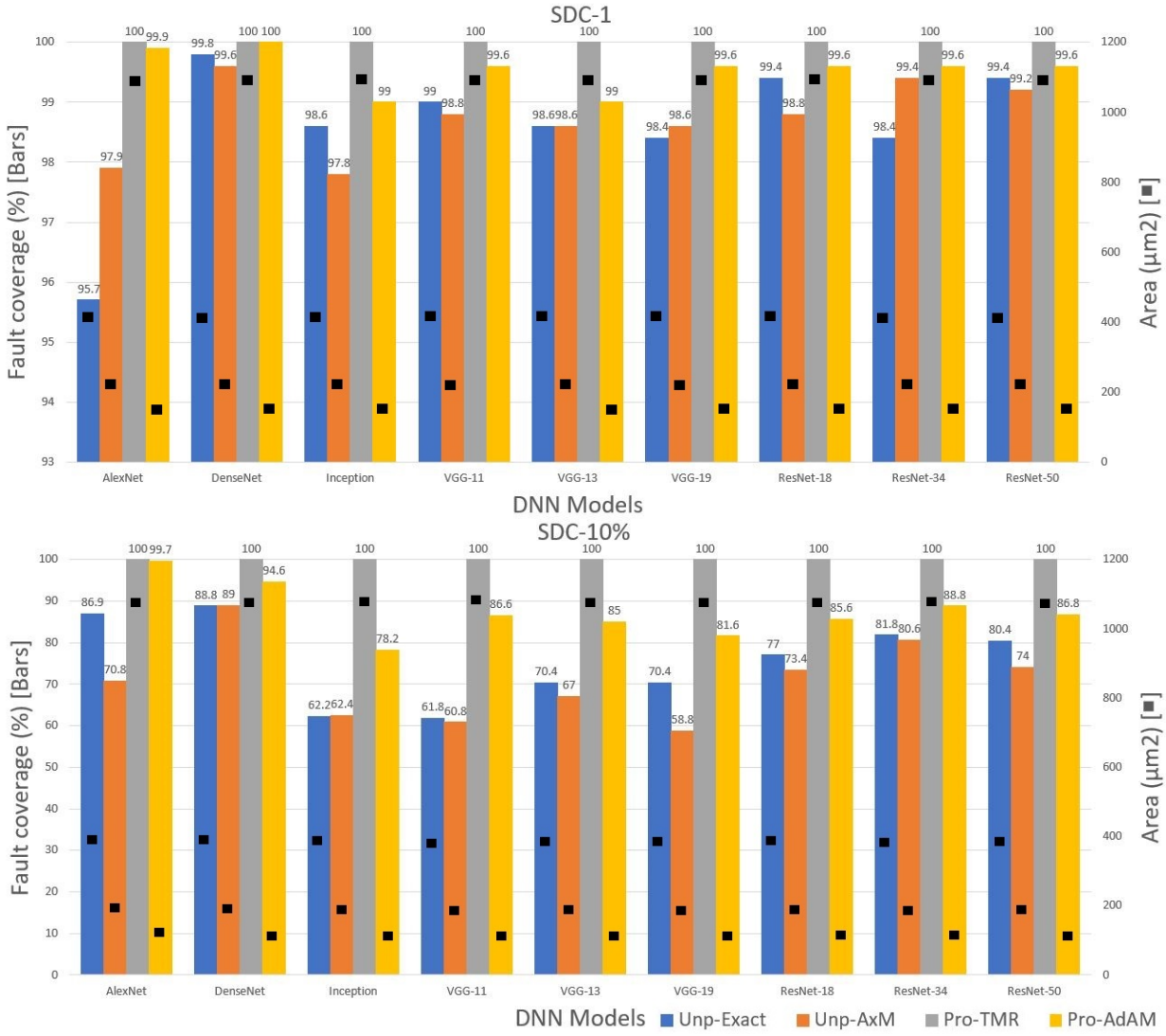


Fig. 7: Hardware efficiency (area) and fault resilience (fault coverage considering SDC-1 and SDC-10%) trade-offs in different benchmarks: AlexNet (MNIST), DenseNet (CIFAR), Inception (CIFAR), ResNet-18 (CIFAR), ResNet-34 (CIFAR), ResNet-50 (CIFAR), VGG-11 (CIFAR), VGG-13 (CIFAR), VGG-16 (CIFAR). Unp-Exact: unprotected exact multiplier, Unp-AxM: unprotected approximate multiplier, Pro-TMR: exact multiplier protected with TMR, Pro-AdAM: proposed multiplier

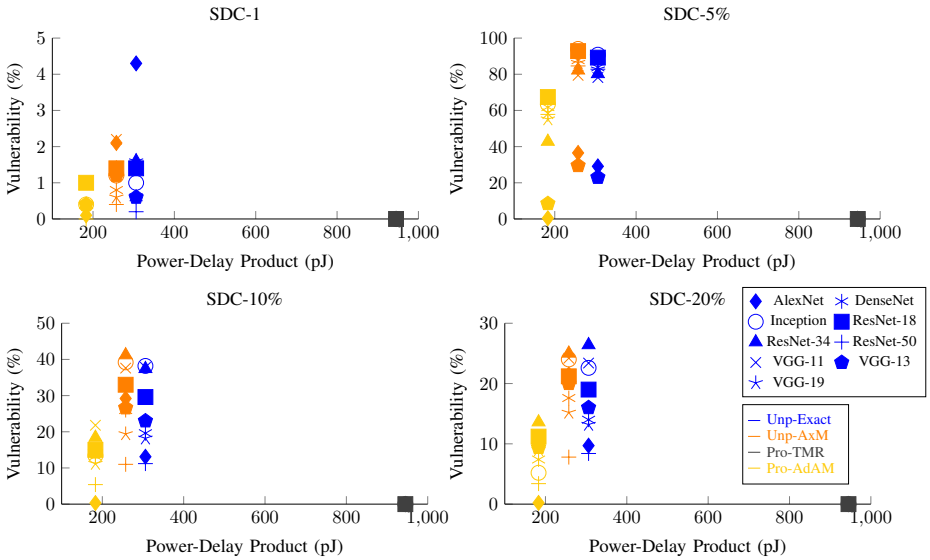


Fig. 8: PDP and vulnerability tradeoffs (considering different SDCs) in different benchmarks: AlexNet, DenseNet, Inception, ResNet-18, ResNet-34, ResNet-50, VGG-11, VGG-13, VGG-16. Unp-exact: unprotected exact multiplier, Unp-AxM: unprotected approximated multiplier, Pro-TMR: exact multiplier protected with TMR, Pro-AdAM: proposed multiplier

lower-order bits (which can be interpreted as errors caused by approximate multipliers), do not always degrade accuracy. This phenomenon aligns with the broader understanding that DNNs are not strictly deterministic and can potentially exhibit more accuracy under specific fault conditions. Thus, the accuracy gains with AdAM can be seen conceptually as part of this broader context of DNN fault tolerance and behavior.

### *D. Reliability analysis*

To showcase the impact of the AdAM multiplier on reliability and performance trade-offs, the fault injection simulations are performed on a variety of 8-bit convolutional neural network architectures including AlexNet, DenseNet, Inception, VGG-11, VGG-13, VGG-19, ResNet-18, ResNet-34, and ResNet-50 with four different configurations: unprotected exact multipliers (Unp-Exact), unprotected approximate multipliers (Unp-AxM), protected exact multipliers with TMR (Pro-TMR), and protected approximate multiplier with AdAM (Pro-AdAM). The DNN reliability is evaluated by comparing the output probability vector of the golden run (i.e. the DNN that behaves as expected, without faults) and the faulty run (i.e. the DNN that includes the fault). The output probability vector represents the list of possible classes with their associated confidence scores predicted by the DNN. The SDC rate is defined as the proportion of faults that caused misclassification in comparison with the golden model. Since in DNNs, there is often not a single correct output, but a list of ranked outputs, each with a confidence score [43], we need to use new criteria to determine what constitutes an SDC for a DNN application. Therefore, we consider four types of SDCs as follows:

- **SDC-1:** The top-ranked element predicted by the DNN is different from that predicted by its fault-free execution. This is critical to assess because the top-ranked element is what is typically used for downstream processing. This criterion directly compares the class with the highest confidence score in the output probability vector of the faulty run to that of the golden run. If the top-ranked class differs between the two runs, this is counted as an SDC-1.
- **SDC-5%:** More than 5% of variation in the top-ranked output confidence score compared to the golden model. This criterion focuses on the confidence score of the top-ranked class in the output probability vector. If the confidence score deviates by more than 5% from the golden run, it is counted as an SDC-5%.
- **SDC-10%:** More than 10% of variation in the top-ranked output confidence score compared to the golden model. Similarly, this criterion looks for deviations in the confidence score of the top-ranked class by more than 10% from the golden run.
- **SDC-20%:** More than 20% of variation in the top-ranked output confidence score compared to the golden model. This criterion identifies faults where the confidence score of the top-ranked class in the output probability vector deviates by more than 20% from the golden run.

All four criteria compare values of the output probability vector of the faulty run to that of the golden run. SDC-1 assesses the correctness of the top-ranked class, while SDC-5%, SDC-10%, and SDC-20% evaluate how much the confidence score of the top-ranked class changes in the faulty run relative to the golden run.

To quantify the effectiveness of each method in mitigating faults, *fault coverage* metric is used. It is defined as (100 − SDC value) representing the percentage of faults that are correctly handled (i.e., do not result in silent data

corruption) by each method.

Fig. 7 demonstrates the fault tolerance comparison and reliability improvement (for SDC-1 and SDC-10% as two examples) of different networks by using the protected approximate multiplier proposed in this work compared to the unprotected exact and approximated networks, and protected networks using TMR. As illustrated, TMR has 100% of protection, but it also requires about 200% of area overhead. Different from TMR, in our technique we introduce a high-reliability improvement without introducing hardware overhead. The results for reliability improvement considering SDC-5% and SDC-20% are reported in Tables VII and VIII. Since the main objective of the proposed multiplier is to have the best trade-off in PDP and vulnerability, Fig. 8 illustrate these comparisons based on different vulnerability metrics (SDC-10, SDC-5%, SDC-10% and SDC-20%). In these charts, the closer to the origins (0,0), the higher the cost-efficiency of the fault tolerance, i.e. less vulnerability and less PDP. As shown, TMR is a less efficient solution for edge AI applications because of its high PDP, while the proposed method (AdAM) is the closest to the origin.

## V. Conclusion

In this paper, we propose an architecture of a novel adaptive fault-tolerant approximate multiplier tailored for ASIC-based DNN accelerators. AdAM employs an adaptive adder that relies on an unconventional use of input Leading One Detector (LOD) values for fault detection by optimizing unutilized adder resources. A gate-level optimized LOD design is also proposed to improve the hardware performance as part of the adaptive multiplier. The proposed architecture uses a lightweight fault mitigation technique that sets the detected faulty bits to zero.

It is demonstrated that the proposed architecture enables a multiplication with a reliability level close to the multipliers protected by TMR while at the same time utilizing 2.74× less area and with 39.06% less power-delay product compared to the exact multiplier.

## VI. Acknowledgement

This work was supported in part by the EU through the project "TAICHIP" grant #101160182, by the Estonian Research Council grant PUT PRG1467 "CRASHLESS" and by Estonian-French PARROT project "EnTrustED".

## References

[1] 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 Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

[2] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A comprehensive survey on model quantization for deep neural networks in image classification," *ACM Transactions on Intelligent Systems and Technology*, vol. 14, no. 6, pp. 1–50, 2023.

[3] M. Nourazar, V. Rashtchi, A. Azarpeyvand, and F. Merrikh-Bayat, "Code acceleration using memristor-based approximate matrix multiplier: Application to convolutional neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2684–2695, 2018.

[4] M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjödin, and B. Lisper, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2023, pp. 1–8.

[5] M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. Della Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo *et al.*, "Special session: Approximation and fault resiliency of dnn accelerators," in *2023 IEEE 41st VLSI Test Symposium (VTS)*. IEEE, 2023, pp. 1–10.

[6] P. Nayak, D. Zhang, and S. Chai, "Bit efficient quantization for deep neural networks," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 2019, pp. 52–56.

[7] M. Taheri, N. Cherezova, M. S. Ansari, M. Jenihhin, A. Mahani, M. Daneshtalab, and J. Raik, "Exploration of activation fault reliability in quantized systolic array-based dnn accelerators," *arXiv preprint arXiv:2401.09509*, 2024.

[8] M. Taheri, N. Cherezova, S. Nazari, A. Rafiq, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, "Adam: Adaptive fault-tolerant approximate multiplier for edge dnn accelerators," *In press, ETS 2024, arXiv:2403.02936*, 2024.

[9] M. Andjelkovic, O. Schrape, A. Breitenreiter, and M. Krstic, "Set and seu hardened clock gating cell," in *2023 38th Conference on Design of Circuits and Integrated Systems (DCIS)*, 2023, pp. 1–6.

[10] P. Rech, "Artificial neural networks for space and safety-critical applications: Reliability issues and potential solutions," *IEEE Transactions on Nuclear Science*, 2024.

[11] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 23, no. 6, pp. 1180–1184, 2014.

[12] S. Hashemi, R. I. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 418–425.

[13] M. Taheri, H. Zandevakili, and A. Mahani, "A high-performance memristor-based smith-waterman dna sequence alignment using fpni structure," *Journal of Applied Research in Electrical Engineering*, vol. 1, no. 1, pp. 59–68, 2022.

[14] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Tosam: An energy-efficient truncation-and rounding-based scalable approximate multiplier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 5, pp. 1161–1173, 2019.

[15] L. Sayadi, S. Timarchi, and A. Sheikh-Akbari, "Two efficient approximate unsigned multipliers by developing new configuration for approximate 4: 2 compressors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 4, pp. 1649–1659, 2023.

[16] M. H. Haider and S.-B. Ko, "Booth encoding based energy efficient multipliers for deep learning systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023.

[17] E. Farahmand, A. Mahani, B. Ghavami, M. A. Hanif, and M. Shafique, "scaletrim: Scalable truncation-based integer approximate multiplier with linearization and compensation," *arXiv preprint arXiv:2303.02495*, 2023.

[18] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[19] G. Alsuhli, V. Sakellariou, H. Saleh, M. Al-Qutayri, B. Mohammad, and T. Stouraitis, *Number Systems for Deep Neural Network Architectures*. Springer, 2023.

[20] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.

[21] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–36, 2022.

[22] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. 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*, 2017, pp. 1–12.

[23] G. Rodrigues, F. L. Kastensmidt, and A. Bosio, "Survey on approximate computing and its intrinsic fault tolerance," *Electronics 2020, Vol. 9, Page 557*, vol. 9, p. 557, 3 2020.

[24] A. Bosio, S. D. Carlo, P. Girard, A. Ruospo, E. Sanchez, A. Savino, L. Sekanina, M. Traiola, Z. Vašíček, and A. Virazel, *Design, Verification, Test, and In-Field Implications of Approximate Digital Integrated Circuits*. Springer International Publishing, 2022, pp. 349–385.

[25] M. Traiola, B. Deveautour, A. Bosio, P. Girard, and A. Virazel, *Test and Reliability of Approximate Hardware. Approximate Computing*. Springer International Publishing, 2022, pp. 233–266.

[26] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.

[27] A. Siddique and K. A. Hoque, "Exposing reliability degradation and mitigation in approximate dnns under permanent faults," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 4, pp. 555–566, 2023.

[28] S. Saeedi, A. Carpegna, A. Savino, and S. Di Carlo, "Prediction of the impact of approximate computing on spiking neural networks via interval arithmetic," in *2022 IEEE 23rd Latin American Test Symposium (LATS)*. IEEE, 2022, pp. 1–6.

[29] A. Ruospo, E. Sanchez, L. M. Luza, L. Dilillo, M. Traiola, and A. Bosio, "A survey on deep learning resilience assessment methodologies," *Computer*, vol. 56, no. 2, pp. 57–66, 2023.

[30] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 258–261.

[31] A. Siddique, K. Basu, and K. A. Hoque, "Exploring fault-energy trade-offs in approximate dnn hardware accelerators," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2021, pp. 343–348.

[32] S. Mittal, "A survey on modeling and improving reliability of dnn algorithms and accelerators," *Journal of Systems Architecture*, vol. 104, p. 101689, 2020.

[33] G. Ammes, G. B. Manske, P. F. Butzen, A. I. Reis, and R. P. Ribas, "Atmr design by construction based on two-level als," in *2023 36th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE, 2023, pp. 1–6.

[34] T. Arifeen, A. S. Hassan, and J.-A. Lee, "Approximate triple modular redundancy: A survey," *IEEE Access*, vol. 8, pp. 139 851–139 867, 2020.

[35] B. Deveautour, M. Traiola, A. Virazel, and P. Girard, "Qamr: an approximation-based fully reliable tmr alternative for area overhead reduction," in *2020 IEEE European test symposium (ETS)*. IEEE, 2020, pp. 1–6.

[36] M. Traiola, J. Echavarria, A. Bosio, J. Teich, and I. O'Connor, "Design space exploration of approximation-based quadruple modular redundancy circuits," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[37] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *International conference on machine learning*. PMLR, 2015, pp. 1613–1622.

[38] D. Danopoulos, G. Zervakis, K. Siozios, D. Soudris, and J. Henkel, "Adapt: Fast emulation of approximate dnn accelerators in pytorch," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[39] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 502–506.

[40] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "Roba multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 2, pp. 393–401, 2016.

[41] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Letam: A low energy truncation-based approximate multiplier," *Computers & Electrical Engineering*, vol. 63, pp. 1–17, 2017.

[42] A. Ruospo, E. Sanchez, M. Traiola, I. O'Connor, and A. Bosio, "Investigating data representation for efficient and reliable convolutional neural networks," *Microprocessors and Microsystems*, vol. 86, p. 104318, 2021.

[43] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. 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*, 2017, pp. 1–12.