



Investigating the effect of approximate multipliers on the resilience of a systolic array DNN accelerator

Salvatore Pappalardo, Ali Piri, Annachiara Ruospo, Ian O'Connor, Bastien Deveautour, Ernesto Sanchez, Alberto Bosio

► To cite this version:

Salvatore Pappalardo, Ali Piri, Annachiara Ruospo, Ian O'Connor, Bastien Deveautour, et al.. Investigating the effect of approximate multipliers on the resilience of a systolic array DNN accelerator. 36th IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT 2023), Oct 2023, Juan-Les-Pins, France. 10.1109/DFT59622.2023.10313535 . hal-04674779

HAL Id: hal-04674779

<https://hal.science/hal-04674779v1>

Submitted on 21 Aug 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Investigating the effect of approximate multipliers on the resilience of a systolic array DNN accelerator

Salvatore Pappalardo¹, Ali Piri¹, Annachiara Ruospo², Ian O'Connor¹,
Bastien Deveautour³, Ernesto Sanchez², Alberto Bosio^{1,3} *Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE
Lyon, INL, UMR5270, 69130 Ecully, France*
²*Politecnico di Torino, Dip. di Automatica e Informatica, Torino, Italy*
Email: ¹{name.surname}@ec-lyon.fr, ²{name.surname}@polito.it, ³{name.surname}@cpe.fr

Abstract—Deep Neural Networks (DNNs) are nowadays extremely popular in different fields of edge and mobile real-time applications such as surveillance, autonomous systems, etc... Energy efficiency and performance are crucial in such context and, for this reason, several DNN hardware accelerators have been designed using Approximate Computing (AxC) techniques. However, other than efficiency, real-time applications used in safety-critical systems (e.g., autonomous car) require a given level of resilience to hardware faults. Indeed, in the literature, many works discussed so far how to assess the resilience of a given hardware accelerator and how to harden it through the insertion of fault-tolerant mechanisms. Fault tolerance is usually achieved by using redundancy that costs in terms of area, power consumption and latency. For the case of DNNs, the redundancy is selectively applied to reduce its cost and protect only the “critical” components. This work aims at proposing a novel approach entirely based on the use of AxC to increase the resilience of DNNs without using redundancy and thus avoiding extra costs. In particular, we studied the impact of AxC multipliers on a systolic array architecture used to accelerate the DNN execution. Preliminary results show that by using an AxC multiplier, it is possible to improve the resilience of almost 10% with better efficiency than the “precise” implementation.

Index Terms—reliability, neural networks, approximate computing, hardware accelerator

I. INTRODUCTION

In the last decades, Deep Neural Networks (DNNs) have revolutionized the field of artificial intelligence and machine learning. They are very powerful, able to learn from a vast amount of data and make complex predictions. In specific tasks, such as image classification, they have demonstrated to be able to surpass human performance [1]. Due to their outstanding computational capabilities, they are employed in many different fields, some of them are labeled as safety-critical. For example, [2] used a neural network to determine the flight regime of an aircraft using temporal segmentation, achieving more than 90% accuracy. Another example can be found in [3], in which the authors proposed a novel approach for lane detection in the context of autonomous drive.

One of the challenges associated with DNNs is their high energy consumption, as well as their high computational power and high amounts of time to complete a single inference. As an example, DNN models used for image processing are

composed by many convolutional layers, and convolution is a really expensive computation [4]. To address this issue, recently, new types of hardware accelerators have been developed. One of the most popular options is given by systolic arrays. They are composed by a grid of Processing Elements (PEs). Each PE performs a Multiply and Accumulate (MAC) operation, meanwhile forwarding the data to the neighbors. This kind of structure is able to process a convolution with $O(n \times m)$ time complexity (supposing a convolution between two matrices with sizes $n \times n$ and $m \times m$). Furthermore, systolic arrays have a higher throughput than GPUs [5] (which are generally used in modern applications) making them more suitable for real-time applications. In addition to that, they can be used in embedded systems since they are less expensive in terms of energy consumption [6]. To further push the energy and performance efficiency, AxC proved to be a very promising approach [7]. The idea behind AxC is that several applications do not really need to be executed on a “precise” and thus “energy-expensive” hardware. AxC aims at reducing the precision of the hardware in order to save energy consumption. Interestingly, reduced precision leads to applications that produce less accurate but still sufficient results while reducing the required energy by orders of magnitude. Such applications are characterized as intrinsically resilient to noise and errors that affect computation (i.e., due to less precise hardware). Inherent resilience is closely related to the application domain and clearly DNNs are perfect candidates thanks to their inherent resilience to noise [8].

On the other hand, even if hardware accelerators for DNNs come with such inherent resilience to noise, recent studies in the literature have shown that hardware accelerators are not always immune to hardware faults. Thus, inference can be significantly affected, leading to DNN prediction failures that are likely to lead to a detrimental effect on the application [9]–[11]. Therefore, ensuring the resilience of hardware accelerators is crucial, especially when they are deployed on real-time safety-critical systems, i.e., systems in which any failure or design error has the potential to lead to loss of life [12].

The classical technique used to improve the resilience is to introduce fault tolerance mechanisms by adding redundancy in the design [13]. The problem of using redundancy, even in the case of selective fault tolerance [14], is the cost in terms of area, power and latency that can be prohibitive, especially

for edge applications. There is a need to **further reduce the cost of fault tolerance** for DNN hardware accelerators.

This research work proposes a novel approach entirely based on the use of AxC to increase the resilience of DNNs without using redundancy and thus avoiding extra costs. In particular, we studied on a case study the impact of AxC multipliers on the resilience of a systolic array architecture used to accelerate the DNN execution. In our case study, a fault injection (FI) campaign is conducted to understand the impact of single bit-flips on the resilience of the network. Preliminary results have shown that using an AxC multiplier, it is possible to improve the resilience of almost 10% with better efficiency than the accurate implementation.

The rest of this paper is structured as follows: in section II a brief overview of existing works is given; then in section III the experimental setup is explained in details; section IV shows the gathered results; section V concludes the article highlighting future directions.

II. RELATED WORKS

Assessing the resilience of DNNs is a deeply investigated problem today. In [15], the authors presented the main DNN reliability assessment methodologies, focusing mainly on FI techniques used to evaluate DNN resilience that can be summarized as **simulation-based** at the software or the hardware level, **platform-based**, and **radiation-based**. Several works have been published so far concerning the use of AxC in the context of fault tolerance. This research area is generally known as Approximate Triple Modular Redundancy (ATMR) [16], applied at the circuit level. The ATMR approach employs three Approximate Integrated Circuits (Ax-ICs) instead of three fully-precise replicas. For a given input, only one AxIC can provide an incorrect answer. However, ATMR fault tolerance capability mainly depends on the voter that has to be modified [17].

A different approach referred to as Quadruple Approximate Modular Redundancy (QAMR) was presented [18]. The QAMR is based on the idea of selectively approximating a subset of the circuit. The final goal is to have four approximate replicas in such a way as to guarantee that, among the four, at least three of them will provide precise results for a given output. The benefit is that the voter does not have to be modified. However, the design cost is higher since the approximation must be carefully identified.

The described works target fault tolerance through masking. Concerning fault detection only, in [19], the authors exploit approximate computing for image processing applications through duplication with comparison (DWC).

The above techniques summarize the use of AxC to reduce the cost of testing and fault detection/tolerance. Another interesting branch of research is investigating how AxC impacts a system's intrinsic reliability, and whether an approximate application is more or less resilient to hardware faults than the precise application. Indeed, AxC exploits the inherent resilience of an application to noise and computing errors. Therefore, an application executed on approximated hardware is less resilient to hardware faults.

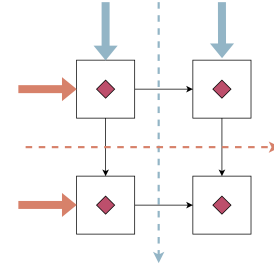


Fig. 1: Simple model of an output stationary systolic array.

In [20], the authors studied the impact of AxC on the reliability of DNN application. As AxC technique, they used the data type and bit-width reduction. In particular, they compared 32-bit floating point versus 16-bit integer data for storing synaptic weights. The DNN was an image classifier based on LeNet-5 topology. The reliability assessment has been done through radiation experiments: the system running the DNN was exposed to neutron beam.

In this paper, we intend to explore the use of a different AxC technique. Instead of weight compression as done in [20], we leverage on approximate multipliers to be used in the hardware accelerator.

III. CASE STUDY

1) *Neural network*: To validate the effectiveness of the technique, the LeNet-5 CNN architecture, trained and tested on MNIST dataset, has been exploited. It is composed of three convolutional layers and four fully connected. The training process was performed by using [21] without any approximation: learning rate started at 0.05, with the decay of 5×10^{-4} every 375(*128) iterations, the momentum was set to 0.9. The final accuracy of the model was equal to 99.05%. We then performed 8-bit quantization through following steps [21].

- 1) all weights are rescaled in the range $[-1.0, 1.0]$ and activations at each layer are rescaled in the range $[-1.0, 1.0]$ for signed outputs and $[0.0, 1.0]$ for unsigned outputs;
- 2) inputs, weights, biases and activations are quantized to the desired n_{bits} by converting $[-1.0, 1.0]$ and $[0.0, 1.0]$ to $[-2^{n_{bits}-1}, 2^{n_{bits}-1}-1]$ and $[0, 2^{n_{bits}-1}-1]$

2) *The systolic array architecture*: The implemented systolic array follows an output-stationary topology [6]. As illustrated in Fig. 1, data flow (blue and orange arrows) from top (north) to bottom (south) and from left (west) to right (east).

Fig. 2 shows a functional diagram of a PE. As above-mentioned, a PE performs a MAC operation; this means that for each clock cycle:

- north and west inputs are multiplied together,
- the result is added to the partial sum register,
- the value of north is put on the south output,
- the value of west is put in the east output.

These operations accumulate a series of multiplications and forward data to the neighbors. This method allows the systolic array to perform a matrix multiplication with linear time-complexity. In our implementation, weights are flowing from

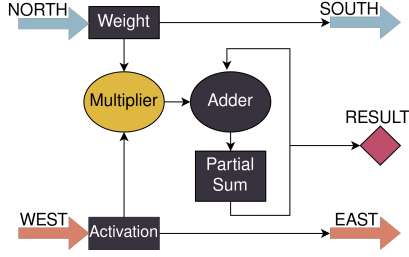


Fig. 2: Functional diagram of a processing element.

north to south and the activations from west to east and are accumulated in the PE itself (square boxes).

To be able to compute an entire convolution in one single pass, the proposed systolic array has the same size of the output of the first convolution in LeNet-5. Since the first layer is composed by 6 convolutions whose output is 28×28 and a single 2D output stationary systolic array can perform a single convolution, our systolic array is a 3D systolic array with the same dimensions of the first layer: $6 \times 28 \times 28$ PEs.

3) *Approximate multipliers*: In this work, different experiments have been conducted with different multipliers. To quantify the effect of inaccuracy on reliability of our systolic array, we replaced the accurate multipliers in each PE with the approximate multipliers presented in EvoApproxLiteLITE¹ [22].

The adopted CNN performs a multiplication between a signed weight and an unsigned activation, thus we needed a mixed-sign multiplier, which is not available in the aforementioned library. As a consequence, we set up the experiments using a 12-bit signed multiplier and simply extending the sign on the operands. Note that this method has the effect of making the approximation even worse: since the multipliers are designed to operate on 12-bits values, using only the least significant 8 bits means reducing the performance of the multipliers. To quantify this effect, we computed the Mean Absolute Error (MAE) and the Worst Case Error (WCE) with the mixed-sign range the networks actually uses. Table I shows the *original* [22] metrics of the multiplier (columns MAE, WCE) and the newly computed metrics that only use our 8bit values ranges. (columns MAE-8, WCE-8).

A. Experimental setup

A high-level implementation made in C language was derived using N2D2 [21]. The network was quantized in order to use int8 values. Specifically, the weights are represented as **signed 8bit integers** values, while the activations were represented as **unsigned 8bit integers**.

Five different experiments exploiting the following approximate multipliers were performed: mul12s_2PT, mul12s_2QH, mul12s_2R5, mul12s_34P, mul12s_2TE. The method used for generating these multipliers is based on decomposing the multiplication and processing each with a combination of approximate circuits [22]. The introduced approximation varies based on the combination used.

¹It is possible to find the used multiplier at the link https://ehw.fit.vutbr.cz/evoapproxlib/?folder=multipliers/12x12_signed

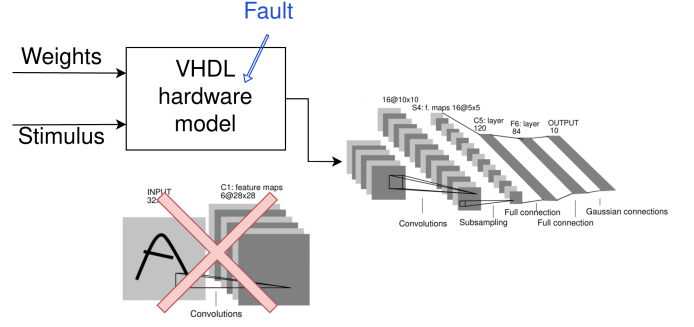


Fig. 3: Fault injection process

TABLE I: Characterization of the approximate multipliers. Every value is a percentage.

Name	MAE	WCE	MAE-8	WCE-8
mul12s_2PT	0.000073	0.00029	0.019	0.0748
mul12s_2QH	0.0031	0.013	0.134	0.514
mul12s_2R5	0.0092	0.037	0.315	1.234
mul12s_34P	0.032	0.17	0.785	3.920
mul12s_2TE	0.19	0.77	6.080	24.417

According to [23], the most critical layer when neurons inputs are faulty is the first one; so, in this work, the experimental efforts were carried out in this layer, while the others are executed in software using the high-level implementation of the network. Differently, in this work, transient faults affecting the registers have been used as fault model. More in details, the performed experiments inject only the weight register of the PEs (corresponding to the block *weight* in figure 2).

The space of possible faults is then 3-dimensional, and its size depends on both the array size and the bit-width: $\text{FaultSpace} = 2 \times K \times M$.

We have a total of M PEs. Each PE has K bits for the input of the weights. Each fault corresponds to an alteration of one bit, forcing its value to either 1 or 0. We call the forced value **polarity**, and we have only two possible values (1 or 0). A statistical FI was configured by assuming a 1% margin of error, 50% probability of a fault resulting in a failure, a cut-off point of 2.58 which corresponds to 99% confidence level.

Accordingly, a FI campaign was performed for every one of the five approximated multipliers. For each campaign, we injected a total of 15'000 single-bit faults, and each injected fault was simulated on 100 different input stimuli (10 images for each class).

The evaluation of the fault is the same as described in [23]. For each input we have the fault-free output \hat{Y} and the faulty output a vector \tilde{Y} . The outputs are vectors of 10 components, each corresponding to the probability of a class. The classification labels \hat{y} and \tilde{y} are obtained as $\hat{y} = \text{argmax}(\hat{Y})$ and $\tilde{y} = \text{argmax}(\tilde{Y})$. For the purpose of the resilience classification, we compare the two vectors \hat{Y} and \tilde{Y} for each pair input-fault as follows:

- we call a fault **masked** when $\hat{Y} = \tilde{Y}$,
- **good** when $\hat{y} = \tilde{y}$ and $\frac{\max(\tilde{Y})}{\max(\hat{Y})} > 1$,

TABLE II: Fault-free classification accuracy after the replacement of the multiplier in the convolution. Bold text shows the best performance.

Multiplier	Accuracy (%)
accurate	99.05
mul12s_2PT	99.08
mul12s_2QH	99.10
mul12s_2R5	99.06
mul12s_34P	98.24
mul12s_2TE	9.80

- **accept** when $\hat{y} = \tilde{y}$ and $0.95 < \frac{\max(\tilde{Y})}{\max(Y)} < 1$,
- **warning** when $\hat{y} = \tilde{y}$ and $\frac{\max(\tilde{Y})}{\max(Y)} < 0.95$,
- **critical** when $\hat{y} \neq \tilde{y}$.

Basically, we check whether the top-1 probability of the injected network $\max(\tilde{Y})$ is greater or smaller than the golden top-1 probability $\max(Y)$ and classify the fault accordingly. Note that **masked** faults do not produce any difference in the output, while only **critical** faults involve misclassification. Finally, we classify as **benign** the faults that fall in the first two categories (masked and good), since not only there is not misclassification, but the confidence of the prediction is equal or higher than the fault-free run. The other three categories are defined as malignant.

In order to fairly evaluate the performances of the FI campaigns, the accuracy of the Neural Network (NN) was evaluated without injecting any fault, substituting the accurate multipliers with the approximate ones. We evaluated the classification accuracy of the network on the entire validation set. To speed up the FI process, i.e., without running architectural-level simulations we used the C code of each multiplier, available in [22].

IV. RESULTS

A. Intrinsic NN robustness

First, the C model of the network was evaluated on its own (i.e., without injecting any fault) to understand the effect of the approximate multipliers. The approximate multiplier replaced the accurate logic, but this was done only for the convolutions, the rest of the computations were accurate.

Table II shows the results in terms of classification accuracy. The accuracy is computed using a validation set of 10'000 input images **without injecting any fault**. Interestingly, the NN withstands the approximation with grace. Noteworthy, the accuracy slightly increases when using approximated multipliers whose error is quite small. This result shows that the introduced error, due to approximation, might even be beneficial for the NN. Furthermore, this result is similar to what shown in [24], in which the authors show that the accuracy increases when using an approximate multiplier and then performs some retraining steps. Interestingly, we find an improvement (although minimal) without retraining the network.

While the best case improvement is only 0.05%, it is a non-negligible result. This is because it corresponds to 500 more input images correctly classified, when compared with the accurate counter-part. Nevertheless, introducing major

TABLE III: Percentage of masked and benign faults per each campaign. Bold text shows the best performance.

multiplier	masked(%)	benign(%)	critical(%)
accurate	64.65	84.18	0.15
mul12s_2PT	63.90	84.20	0.15
mul12s_2QH	38.92	79.79	0.20
mul12s_2R5	26.96	76.31	0.30
mul12s_34P	74.16	88.08	0.37
mul12s_2TE	3.94	14.44	48.59

approximations produces nefarious effects. Expectedly, the worst multiplier introduced an error so big that the accuracy plummeted to less than 10%. The confusion matrix shows that the NN classified every input as a 0.

B. Fault injection

1) *General resilience*: Figure 4 shows the general behaviour of the different configurations. Furthermore, Table III reports the percentage of masked, benign and critical faults per each FI campaign. It is possible to notice that the mul12s_2PT produces the same effect as the accurate multiplier. Increasing the error, multipliers mul12s_2QH and mul12s_2R5 have the effect of decreasing the resilience, since the number of masked and benign faults decrease significantly. On the other hand, mul12s_34P shows a completely different behaviour: the majority of injected faults are masked. Compared to the accurate network, **there is a 9.5% increase in masked faults**, even though the classification accuracy dropped by about 1%. This outcome shows the benefit obtained by using this multiplier in this NN. Finally, multiplier mul12s_2TE is not reliable in any measure, “misclassifying” the majority of the inputs; actually, about 50% of the inputs were correctly classified when compared with the ground truth, but its performance is unacceptable anyway. Similar results were found by the authors in [25], who show that introducing approximation may decrease the general accuracy of the network but increase the resilience of the network dramatically.

2) *Correlation between injected bit and resilience*: Figure 5 shows the number of malignant faults (normalized by the number of injections) with respect to the injected bit. The yellow bar represents the total number of FIs, while the blue bar is only referred to 1-polarity injections. The superimposed text indicates the percentage of 1-polarity faults. Bit 1 is the most significant, while bit 8 is the least significant. The weights are encoded as signed 8-bit integers.

The first four (from left to right) graphs show the same trend. The first bit is the sign bit, and it is less critical than the second one. The second to the last bits show a decreasing criticality, which is expected since the least significant bits have smaller values. Furthermore, 1-polarity injections are more critical for every bit but the first. This behaviour is due to the regularization of the weights, which squeezes all the weights around zero, making it less common to have ones in the higher bits. Finally, the first bit is most critical to 0-polarity injections. This depends on the data distribution: the majority of injections did not change the actual value, writing a 1 where there already was a 1, but in general, we see that 0-polarity

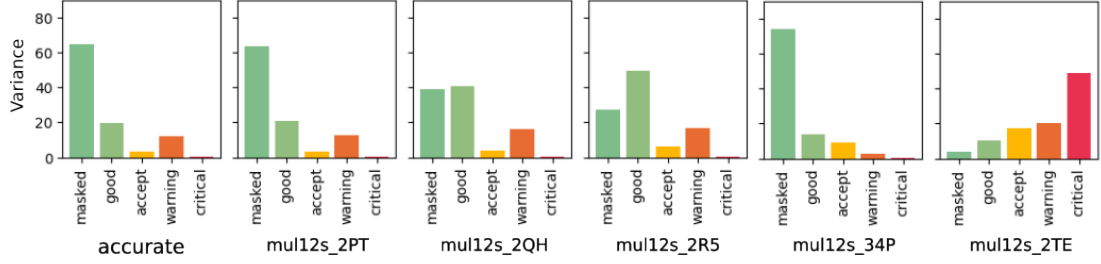


Fig. 4: General resilience for each experiment. Each figure describes the distribution of the fault categories per experiment.

faults are more critical, representing the 73% of malignant outcomes.

The most interesting results are produced by the multiplier `mul12s_34P`. In this case, FIs in the three Least Significant Bits (LSBs) seldom resulted in malignant faults (less than 1% of the injections per each bit). Furthermore, 90% of the malignant faults were produced by 1-polarity injections in bits 2 to 5. This behaviour is similar to the other multipliers, but shows that the network with this multiplier is extremely critical to faults that increase the magnitude of the value rather than faults that decreases it. This datum can be used to deploy reliable hardware by simply masking such a fault with a 0 value.

C. Reliable approximate multiplier

Multiplier `mul12s_34P` produced a very interesting result, since more than **74% of the injections produced masked faults**. This is due to the small variation of the output of the multiplier when changing the three LSBs. In order to show this concept, we introduced a new local metric. We gathered the approximate multipliers, and we computed all the possible combinations storing both inputs and outputs. Then, we grouped the data so that for each bucket, only the three LSBs of the input of the weight varied. Finally, for each bucket, we computed the mean and the variance of the outputs. Figure 6 shows the variance plotted for each multiplier for each input. On the x axis there are the different combinations of inputs allowed (excluding the three LSBs obviously), while on the y axis there is the variance. The red lines and the superimposed text show how many points are strictly under variance 1, that is, inputs that produce the same output when varying the three LSBs.

The first multiplier is the accurate one. It shows quadratic behavior. Note that the outputs generated varying only one of the two inputs have the same distance, so the shown parabola is composed of many horizontal segments. This behavior can be explained as follows. The multiplier has two inputs a and w . The output is computed as $o = w \times a$. If we name the outputs based on input w (thus fixing the value of a), we have $o_i = w_i \times a$. For each pair o_i and o_j there is a difference which can be computed with the formula $d_{ij} = |o_i - o_j| = |i - j| \times a$, simply because a is fixed. With this concept in mind, is simple to understand that the variance of successive buckets with the same value of a has the same variance. Furthermore, the variance decreases as a decreases, since the spreading among values is proportional to a . As is to be expected, multipliers

`mul12s_PT`, `mul12s_QH` and `mul12s_R5` show the same quadratic behavior although noisier. Multiplier `mul12s_TE` shows a completely different behavior: the outputs vary very little in really big ranges. This datum is a confirmation of MAE and MRE values.

Multiplier `mul12s_34P` shows a very interesting behavior since all the points have 0 variance. This graph shows that varying the LSBs has a no effect in many cases.

V. CONCLUSIONS

The results obtained in these FI campaigns suggest that an approximate multiplier can be used for NN computations making the whole NN more resilient. Furthermore, we can identify two properties that the multiplier should have in order to increase the resilience of the NN:

- the approximate output must be always somewhat similar to the accurate computation (MAE and MRE are good metrics in this regard),
- the output value should vary very little (or nothing) when changing the LSBs of the inputs.

The first property is fundamental for approximate computing. In general, the smaller the error of the approximate multiplier, the better the multiplier is. The second property is more interesting and can be used to build new metrics for quantifying the resilience of a NN. Nevertheless, our method does not investigate the effect of all the input bits of the multiplier, so more effort has to be put in the investigation of such phenomena. This method could also be used, in principle, for a general circuit that is put in a NN. For this reason, the next direction of investigation will be building a proper metrics to quantify the resilience of circuits.

In general, approximate multipliers can be really useful in the context of NNs. They use less energy than accurate circuits and, as seen, may provide more accuracy. They can also be used to increase the resilience at the cost of some accuracy loss (which may or may not be acceptable depending on the application). We showed that some multipliers are best suited for increasing the resilience of the NN. Specifically, one of the multipliers increased the number of masked faults by an outstanding 9.3% of masked faults. Furthermore, we provided a simple method to understand why this specific multiplier boosted the performance so much, which also gave some useful insights for the design of resilient circuits.

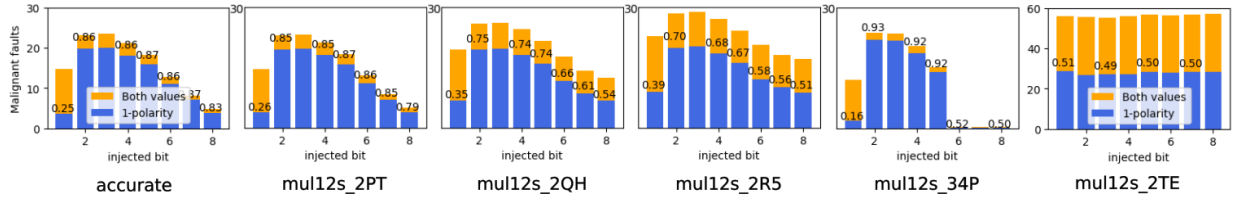


Fig. 5: Number of malignant faults per bit normalized to the number of injections.

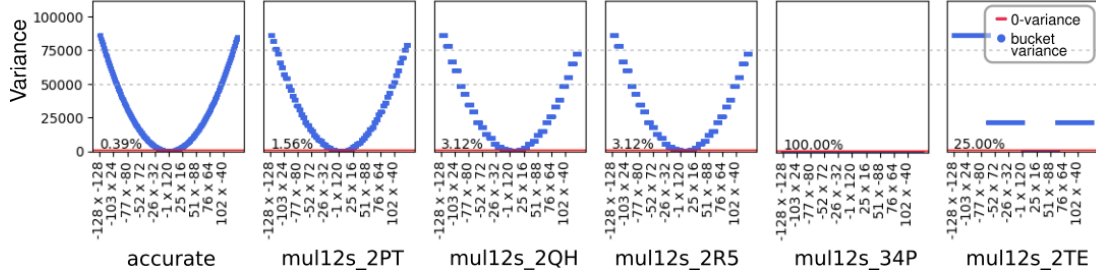


Fig. 6: Variance of the output per each input combination varying on the three LSBs. The y-axis is the value of the variance. The red-line and the superimposed percentage show how many points have variance strictly less than 1.

ACKNOWLEDGMENT

This work has been funded by the RE-TRUSTING project, ANR-21-CE24-0015; the APROPOS project in the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956090; and the FAIR - Future Artificial Intelligence Research and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3 – D.D. 1555 11/10/2022, PE00000013). This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

REFERENCES

- [1] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015.
- [2] J. Wu *et al.*, "Aircraft flight regime recognition with deep temporal segmentation neural network," *Engineering Applications of Artificial Intelligence*, vol. 120, p. 105840, 2023.
- [3] S. Ghanem, P. Kanungo, G. Panda, and P. Parwekar, "An improved and low-complexity neural network model for curved lane detection of autonomous driving system," *Soft Computing*, vol. 27, no. 1, pp. 493–504, 2023.
- [4] G. Desoli *et al.*, "14.1 a 2.9 tops/w deep convolutional neural network soc in fd-soi 28nm for intelligent embedded systems," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 238–239, IEEE, 2017.
- [5] M. Andersch *et al.*, "On latency in gpu throughput microarchitectures," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 169–170, IEEE, 2015.
- [6] V. Sze *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
- [7] A. Bosio, D. Ménard, and O. Sentieys, eds., *Approximate Computing Techniques*. Springer International Publishing, 2022.
- [8] E. Dupuis, S. Filip, O. Sentieys, D. Novo, I. O'Connor, and A. Bosio, "Approximations in deep learning," in *Approximate Computing Techniques*, pp. 467–512, Springer International Publishing, 2022.
- [9] A. Lotfi *et al.*, "Resiliency of automotive object detection networks on gpu architectures," in *2019 IEEE International Test Conference (ITC)*, pp. 1–9, IEEE, 2019.
- [10] B. Salami, O. S. Unsal, and A. C. Kestelman, "On the resilience of rtl nn accelerators: Fault characterization and mitigation," in *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 322–329, IEEE, 2018.
- [11] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, "A reliability analysis of a deep neural network," in *2019 IEEE Latin American Test Symposium (LATS)*, pp. 1–6, IEEE, 2019.
- [12] J. C. Knight, "Safety critical systems: challenges and directions," in *Proceedings of the 24th international conference on software engineering*, pp. 547–550, 2002.
- [13] G. Di Natale *et al.*, *Cross-layer reliability of computing systems*. iet-the institution of engineering and technology, 2020.
- [14] A. Ruospo *et al.*, "Selective hardening of critical neurons in deep neural networks," in *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 136–141, IEEE, 2022.
- [15] A. Ruospo *et al.*, "A survey on deep learning resilience assessment methodologies," *Computer*, vol. 56, no. 2, pp. 57–66, 2023.
- [16] B. D. Sierawski, B. L. Bhuvu, and L. W. Massengill, "Reducing soft error rate in logic circuits through approximate logic functions," *IEEE transactions on nuclear science*, vol. 53, no. 6, pp. 3417–3421, 2006.
- [17] G. S. Rodrigues *et al.*, "Approximate tnr based on successive approximation to protect against multiple bit upset in microprocessors," in *2018 18th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, pp. 1–5, IEEE, 2018.
- [18] M. Traiola *et al.*, "Test and reliability of approximate hardware," in *Approximate Computing*, pp. 233–266, Springer, 2022.
- [19] M. Biasioli *et al.*, "Approximation-based fault tolerance in image processing applications," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 648–661, 2021.
- [20] L. M. Luza *et al.*, "Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 4, pp. 1867–1882, 2021.
- [21] CEA-LIST, "N2D2." [Online]. Available: <https://github.com/CEA-LIST/N2D2>.
- [22] V. Mrazek *et al.*, "Scalable construction of approximate multipliers with formally guaranteed worst case error," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 11, pp. 2572–2576, 2018.
- [23] A. Ruospo *et al.*, "Investigating data representation for efficient and reliable convolutional neural networks," *Microprocessors and Microsystems*, vol. 86, p. 104318, 2021.
- [24] M. S. Ansari *et al.*, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 317–328, 2019.
- [25] M. Taheri *et al.*, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pp. 1–8, IEEE, 2023.