Fault Tolerance in Quantized and Pruned Convolutional Neural Networks

Wilfread Guillemé, Angeliki Kritikakou, Youri Helen, Cédric Killian, Daniel Chillet

Fault Tolerance in Quantized and Pruned Convolutional Neural Networks

Wilfread Guillemé*, Angeliki Kritikakou*§, Youri Helen‖, Cédric Killian‡ and Daniel Chillet*

*Univ. Rennes, Inria/IRISA, §IUF, ‖DGA MI, ‡Université Jean Monnet Saint-Etienne, CNRS, Institut d'Optique Graduate School, Laboratoire Hubert Curien UMR 5516, F-42023, SAINT-ETIENNE, France

***Abstract*—Convolutional Neural Networks (CNN), particularly those used in critical applications, such as autonomous driving, medical systems, and aerospace, require high reliability. While these algorithms exhibit inherent resilience, they remain susceptible to Single-Event Effects (SEE) occurring at the hardware and impacting the model execution. These effects, usually induced by interactions with radiation particles, can lead to errors in electronic components, potentially causing incorrect inferences and increasing the risk of mispredictions. Meanwhile, quantization and pruning are widely employed to reduce the hardware footprint of CNN models, facilitating their deployment on embedded systems. Even when the models are reduced, CNN remain too large for an exhaustive fault injection campaign to assess their resilience. To address these challenges, we propose SFI4NN, a Statistical Fault Injection (SFI) framework specifically designed to evaluate the fault sensitivity of fixed-point quantized and pruned CNN architectures. Furthermore, we analyze the model resilience as a function of the pruning rate, showing that CNN sensitivity increases as pruning becomes more aggressive. The obtained results enable the development of hardware hardening strategies with reduced costs that are tailored to the reliability requirements of targeted applications. Experimental results demonstrate a 96% improvement in resilience, with minimal hardware overhead compared to conventional hardening techniques such as triplication.**

***Index Terms*—CNN, Fault Tolerance, Mitigation, SEU, TMR.**

I. Introduction

Convolutional Neural Networks (CNN) are now deployed in many fields, including autonomous vehicle driving, medical systems, and aerospace. Their reliability is crucial in these critical domains, even though they are inherently resilient. Single-Event Effects (SEE) can occur due to interactions with radiation particles, leading to errors in electronic components during inference. When a particle strikes a memory element, such as a D flip-flop, its value may change leading to Single-Event Upset (SEU). In the presence of such faults, AI algorithms can produce erroneous inferences, increasing the risk of incorrect predictions. This highlights the need to develop hardware hardening solutions to mitigate these bit-flips and improve the CNN reliability. However, deploying CNNs on embedded systems poses challenges related to integration, energy consumption, and execution speed. To address these issues, several techniques have been proposed. One such technique is Quantization-Aware Training (QAT), which is applied during the training phase to preserve prediction accuracy while optimizing hardware usage. CNNs are typically trained using a 32-bit floating-point format. Replacing it with a lower-precision fixed-point format can significantly reduce resource requirements. This format also simplifies mathematical operations, making hardware implementation more efficient than floating-point. The Brevitas library [1] is particularly well-suited for implementing QAT. Another optimization is weight pruning, which removes the least significant weights to reduce the number of operations during inference, especially by skipping multiplications involving near-zero values. This technique accelerates inference, lowers energy consumption, and reduces memory usage. A common way to promote sparsity is by applying $L1$ regularization during training. The Prune library [2] supports this method, resulting in lighter and faster models while maintaining acceptable accuracy.

To assess CNN resilience against SEUs, Statistical Fault Injection (SFI) [3] is commonly used, as exhaustive fault injection is infeasible. Several works have studied the resilience of quantized and pruned CNNs [4, 5, 6]. However, a flexible framework combining fault injection for both quantized and pruned models, including intermediate data, is still lacking. To address this, we propose SFI4NN, a tool for analyzing resilience of (i) quantized CNNs and (ii) pruned models with various pruning rates, targeting faults in weights and intermediate activations. Our framework simulates bit-flips, considering their direction ($0 \rightarrow 1$ or $1 \rightarrow 0$), location (layer and bit), to better understand their impact on predictions. We apply these results to a hardware protection solution lighter than triplication and more cost-effective: VANDOR [7], a Voting Block based on AND/OR gates that handles fault directionality on CNN parameters and data. As a case study, we use the SFI4NN framework to carry out a detailed reliability analysis on AlexNet, identify the most vulnerable layers and bit positions, and evaluate the effectiveness of the VANDOR-based protection approach at different pruning rates.

The remainder of this paper is organized as follows. Section II reviews the related work on CNN fault resilience. Section III describes the proposed SFI4NN fault injection framework, along with the VANDOR hardware hardening solution. Section IV presents and discusses the experimental results: we first describe the CNN architecture and dataset used, then evaluate the fault sensitivity of a quantized model, and finally analyze the impact of different pruning rates. Section V concludes the paper and discusses the effectiveness of the proposed methods.

979-8-3315-3334-2/25/$31.00 ©2025 IEEE

II. Related Work

The CNN resilience is an active research area, where approaches focus on how to evaluate the models' resilience and to develop hardware solutions to improve it. Regarding the resilience assessment of CNN, existing studies mainly evaluate the resilience with respect to faults occurring in model parameters [8]. However, intermediate data, temporarily stored during inference, is also susceptible to faults. This is particularly true for CNN hardware accelerators, which process the models layer by layer [9]. As neural networks grow in size, approaches are required to deal with scalability issues. A SFI method is proposed that limits the number of faults tested per layer and bit, thus lowering the computational load compared to exhaustive methods, leading to efficient evaluation of CNN robustness while significantly reducing simulation time [3]. However, this study focuses only on floating-point representations. Some works explore the vulnerabilities of different data representations. In [10], fixed-point and floating-point formats are compared. In fixed-point representation, the most critical bits are the ones with higher significance, while in floating-point representation, the exponent bits are the most sensitive. Fault injection with machine learning is combined to identify the critical parameters of models using floating-point and fixed-point formats [11]. Furthermore, some works have shown that there are asymmetries in the resilience of the model that are related to the direction of bit-flips [12, 13]. However, their precise impact on CNN classification, particularly in combination with quantization and pruning, still requires further investigation. To complement existing work, we introduce SFI4NN, a fault injection framework for evaluating the resilience of CNNs under different quantization levels and pruning ratios. It considers faults in both network parameters and intermediate activations, and identifies the most sensitive parts of the model for targeted protection.

Classical hardware fault protection approaches generally rely on hardware or informational redundancy. Due to the large size of CNN models, selective fault tolerance is typically used. For instance, triplicating only the Most Significant Bits (MSB) of the parameters [10] and Error Correction Codes (ECC) to protect only the critical parameters and bits [11]. Other strategies exist, such as ApFT [14], which protects a systolic array dataflow architecture by duplicating convolution filters while optimizing the architecture. Other approaches remove the faulty elements. For instance, MOZART+ [15], integrated into stationary output data flow architectures, uses multiplexing to disable defective processing elements while maintaining the overall system operation. Last, method exist that limit the range of the observed values. For instance, FT-ClipAct [16] introduces a threshold mechanism to limit abnormally high weight values to better manage faults at the memory level. In contrast, VANDOR uses duplication to detect faults and applies a more flexible correction method. Instead of removing faulty elements or strictly limiting values, VANDOR selectively moves faulty outputs toward zero, providing a balanced approach between fault tolerance and performance.

III. Proposed Methodology

This section introduces the SFI4NN fault injection framework, designed to evaluate the resilience of quantized and pruned CNN models. It then describes the VANDOR hardware solution, which aims to improve their fault tolerance.

A. SFI4NN: Fault Injection Framework

Exhaustive fault injection requires fault injection at every bit position across all layers of the network, testing each input, which quickly becomes computationally infeasible for complex models. Therefore, statistical approach [17] is typically used to assess the resilience of a CNN model. SFI employs sampling techniques to estimate the impact of faults, while significantly reducing the number of fault injections, without compromising the accuracy of the results. Different layers of a CNN behave differently due to faults, and MSBs have a higher impact on prediction errors. Thus, an SFI approach that takes this information into account, such as the one presented in [3], is particularly effective. This method focuses fault injections on smaller layers and targets MSBs more frequently, as they have a greater influence on the network's behavior. This approach, described by the following equation (1), allows for evaluating the robustness of the CNN against SEUs by defining an error margin and confidence level before the fault injection campaign.

$$n(i,l) = \frac{N(i,l)}{1 + e^2 \cdot \frac{N(i,l)-1}{t^2 \cdot p(i) \cdot (1-p(i))}} \tag{1}$$

$N(i,l)$ depends on the CNN architecture and represents the total number of faults that can be injected into the network, based on the bit position i and layer l. The parameters e and t correspond to the error margin and confidence level defined by the user, respectively. Finally, p denotes the probability of an error occurring at each bit position. It is important to note that the number of injected faults increases for higher-order bits, as they have a greater impact on the network's output and are therefore more likely to cause prediction errors. These probabilities, which determine the fault injection rates based on the layers and bit positions, are computed using equation (2).

$$\forall i \in I \quad p(i) = p_{\text{min}} + \frac{(D(i) - D_{\text{min}})(p_{\text{max}} - p_{\text{min}})}{(D_{\text{max}} - D_{\text{min}})} \tag{2}$$

$D(i)$ represents the distance of a bit-flip on a value based on the selected bit. For a bit-flip on the Least Significant Bit (LSB) of a signed integer, D_{min} is always equal to 1. In contrast, D_{max} depends on the data length I. For example, for an 8-bit signed integer, a bit-flip on the MSB will result in a distance of 128. p_{max} is automatically set to 0.5, while p_{min} depends on the data length, defined by the relation $p_{\text{min}} = \frac{1}{2^{\text{I}}}$.

We developed the SFI4NN framework to adapt the above SFI approach for quantized and pruned CNNs, as illustrated in Figure 1. The user first defines the desired error margin and confidence level. Achieving higher precision in the fault

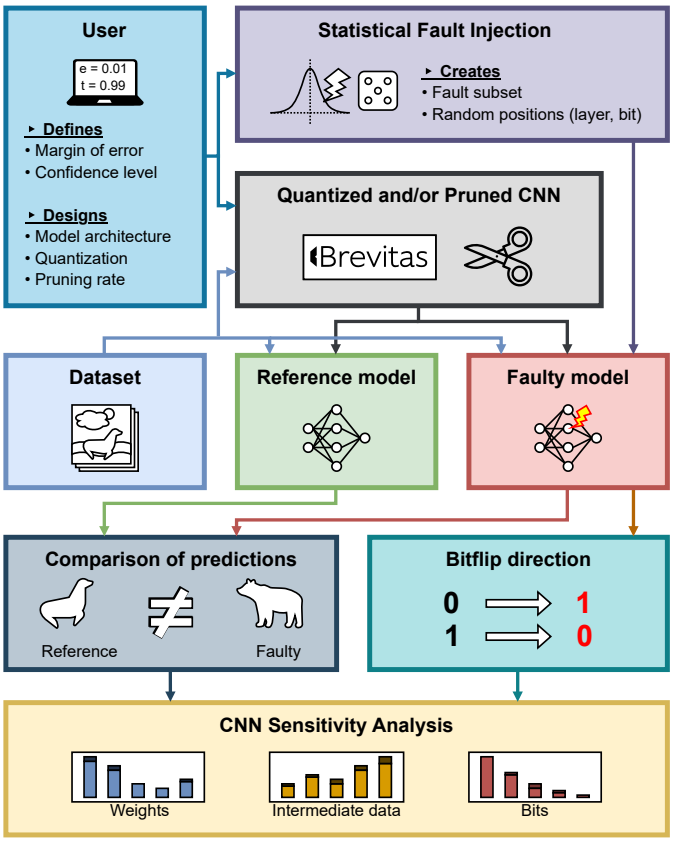


Fig. 1: Statistical fault injection (SFI4NN) framework.

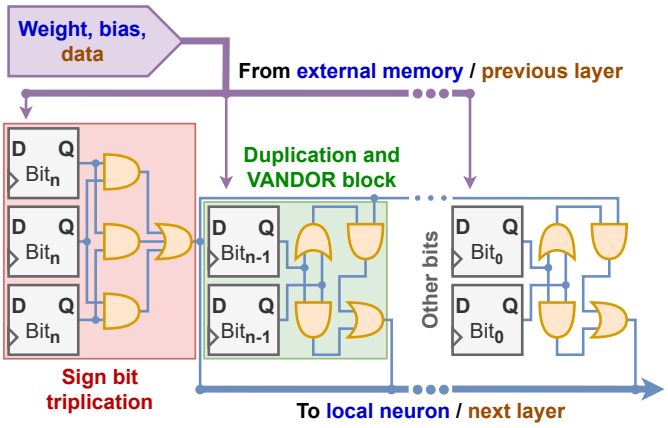


Fig. 2: VANDOR hardware implementation.

TABLE I: VANDOR block truth table.

Sign bit	Sensitive bit	Duplicated bit	VANDOR result
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

tolerance assessment requires injecting a larger number of faults into the CNN, which consequently increases the simulation time. Additionally, the user must specify the CNN hyperparameters, such as the number of layers, channels, and other architectural details. Quantization parameters are defined using the Brevitas framework, while the pruning rate is set with the Prune library. This information is provided to the SFI block, which applies Equation (1) to define a subset of faults at random positions. In parallel, the CNN is trained with QAT using Brevitas, generating two ONNX files: a reference model and a modified version where fault injection is applied. For each test set image, the SFI framework injects faults according to their position, the targeted layer, and the selected bit.

For instance, if a weight's value is 38 in an 8-bit signed integer format (0010 0110) before applying a fault, flipping the third bit changes the value to 46 (0010 1110), corresponding to a distance of $D(i) = 8$. Since the original weight is positive, this bit-flip moves its value away from zero. The faulty model is then loaded with this erroneous value, and an inference with ONNXRuntime [18] is performed. Its prediction is compared to the CNN reference output. A difference between the two predictions indicates an impact on the model output. Repeating this experiment multiple times enables the evaluation of the CNN resiliency, considering layers, weights, intermediate data, and the precise bit-level fault position.

B. VANDOR: Hardware Hardening Solution

The proposed hardware implementation is inspired by the method described in [7]. Known as VANDOR (for Voter block with AND/OR gates), this approach is designed to protect both parameters and intermediate data in CNNs deployed on various hardware accelerators. For this reason, the protection strategy is presented in a general form in this article.

The main idea behind this protection technique is to mitigate the impact of faults by steering erroneous values toward zero whenever an error is detected. This approach leverages the observation that CNNs are generally more resilient to perturbations that result in zero-valued data. Figure 2 illustrates an implementation of the proposed strategy. As shown in the figure, the sign bit of a parameter or data value stored in D flip-flops is triplicated. This redundancy is crucial because the sign bit plays a key role in two's complement representation, directly influencing the behavior of the VANDOR voting block. The remaining bits are simply duplicated and connected to this module. Composed of two AND gates and two OR gates, its behavior is defined by the truth table in Table I. In this table, red cells indicate cases where a fault is detected, i.e., when the sensitive bit and its duplicate differ. The column labeled "VANDOR result" shows the resulting bit value after the voting process. Cells highlighted in green correspond to the chosen bit value. As illustrated, the voting block tends to favor '0' when the sign bit is zero, behaving like an AND gate. Conversely, it favors '1' when the logical state of the sign bit is set to one, acting like an OR gate.

TABLE II: Detailed architecture of the studied AlexNet CNN.

Layer type		Number of data	Output shape	Number of parameters
INPUT	Input-0	3,072	[3, 32, 32]	0
CONV-1	Conv-1	57,600	[64, 30, 30]	1,728
	ReLu-2	57,600	[64, 30, 30]	0
	Pool-3	14,400	[64, 15, 15]	0
CONV-2	Conv-4	32,448	[192, 13, 13]	110,592
	ReLu-5	32,448	[192, 13, 13]	0
	Pool-6	6,912	[192, 6, 6]	0
CONV3	Conv-7	13,824	[384, 6, 6]	663,552
	ReLu-8	13,824	[384, 6, 6]	0
CONV-4	Conv-9	9,216	[256, 6, 6]	884,736
	ReLu-10	9,216	[256, 6, 6]	0
CONV-5	Conv-11	9,216	[256, 6, 6]	589,824
	ReLu-12	9,216	[256, 6, 6]	0
	Pool-13	2,304	[256, 3, 3]	0
FC-1	Flat-14	2,304	[2 304]	0
	Lin-15	4,096	[4 096]	9,437,184
	ReLu-16	4,096	[4 096]	0
FC-2	Lin-17	4,096	[4 096]	16,777,216
	ReLu-18	4,096	[4 096]	0
FC-3	Lin-19	10	[10]	40,960
Total		**289,994**		**28,505,792**

The VANDOR protection strategy is particularly advantageous for its low hardware overhead, making it well-suited for CNN implemented on resource-constrained embedded systems. For instance, protecting an 8-bit data value requires triplicating only the sign bit, while the remaining bits are duplicated. This results in two additional D flip-flops for the sign bit and seven more to duplicate the others bits. Consequently, this solution requires just 9 extra D flip-flops, compared to 16 for a full TMR strategy. Resulting in a significant 44% reduction in D flip-flop usage. In terms of protection efficiency, the proposed method has demonstrated its ability to safeguard CNN weights in over 94% [7] of cases when subjected to SEUs in a LeNet-5 [19] architecture. Several variations of the proposed VANDOR strategy are possible. For enhanced protection, additional high-order bits can be triplicated to improve fault resilience, at the cost of increased hardware overhead. Conversely, omitting protection for low-order bits, which generally have no impact on CNN predictions, can further reduce hardware cost with limited impact on CNN resilience. However, these variants are not explored in this article.

IV. Experiments and Results

This section presents the experimental setup, including the CNN architecture and dataset, followed by analyses of fault sensitivity in both quantized and pruned models.

A. CNN Architecture and Dataset

The CNN architecture used in this study is based on AlexNet [20], which consists of five convolutional (CONV) layers followed by three fully connected (FC) layers, as shown in Table II. Each CONV layer applies convolutional filters, generating, for example, 64 channels in the first layer and 192 channels in the second. MaxPooling operations are applied after some CONV layers to reduce the spatial dimensions while preserving the most important information. The ReLU activation function is used after each layer to introduce non-linearity into the network. The first two FC layers each contain $4,096$ neurons, while the final FC layer has a number of neurons corresponding to the number of classes in the dataset, which is set to 10 in this study. The final classification is determined by the neuron with the highest output value. The AlexNet architecture has about 28.5 million parameters and processes 289,994 intermediate data points per CNN inference.

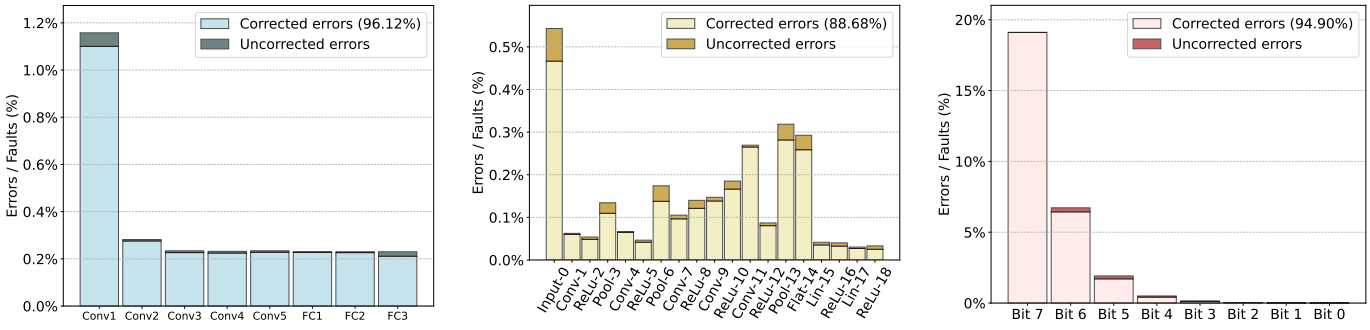
The dataset used in this study is CIFAR-10 [21], consisting of $60,000$ color images divided into 10 distinct classes, with each having a resolution of 32×32 pixels. The training set contains $50,000$ samples, while the test set includes $10,000$.

Since the CNN is designed for execution on an embedded system, the model is quantized using an 8-bit fixed-point format with the Brevitas framework. This method employs a Scale Factor (SF) to map weights to their corresponding signed integer representation, ranging from -128 to $+127$. For instance, if the tool sets the SF value to 0.00390625 for a given layer, this corresponds to shifting the decimal point left by $-\log_2(SF)$ bits. In this particular case, the shift equals 8 bits. QAT optimizes the network weights while maintaining the CNN's accuracy, achieving over 84% in our experiments.

The pruning process follows a normalized $L1$ pruning strategy with a uniform pruning rate applied across all layers. Weights are selected for pruning based on their $L1$ norms, with the smallest values set to zero. Although these pruned weights are masked and no longer affect the model's computations, they remain in the exported ONNX [22] file from Brevitas as zero values. After pruning, a fine-tuning step is performed to help the model regain some of its lost accuracy by re-optimizing the remaining active weights.

B. Fault Sensitivity of a Quantized Model

This first study aims to precisely visualize and quantify the sensitivity of a CNN to faults. Specifically, it evaluates the impact of faults on both the parameter and intermediate data layers, as well as the bit positions within them. For this analysis, an 8-bit quantized model implemented with Brevitas was trained for 100 epochs using the Adam [23] optimizer, achieving a prediction accuracy of 84.03%. At this stage, no pruning has been applied. The fault injection campaign follows the SFI4NN approach, with a 10% error margin and a 90% confidence level. The results are presented in Figure 3. In each subfigure, the X-axis represents either the layer index or the bit position, while the Y-axis indicates the number of errors that resulted in a prediction mismatch between the faulty and the reference model, normalized by the total number of injected faults. Figure 3a illustrates the sensitivity of the layers containing the parameters of the AlexNet model. For this experiment, the $10,000$ test images from the CIFAR-10 dataset were used. This corresponds to 14.69 million faults injected using the SFI4NN tool on the weights of the studied CNN. In total, $51,718$ faults resulted in a prediction mismatch between the faulty and reference models. Among these errors,



(a) Parameters (b) Intermediate data (c) Bit position

Fig. 3: Quantized AlexNet sensitivity and VANDOR protection efficiency.

the majority were located in the first layer. This layer exhibited a sensitivity of over 1.1%, despite containing only $1,728$ weights (as shown in Table II), which is significantly fewer than the other layers in the network. Consequently, a fault in this layer has a proportionally higher impact. In contrast, the remaining layers demonstrate a similar sensitivity level, around 0.2%. Regarding VANDOR's protection performance on model parameters, corrected errors are shown in light blue, representing bit-flips that moved the targeted value away from zero. Conversely, uncorrected errors are depicted in dark blue, corresponding to bit-flips that shifted the value closer to zero. Overall, VANDOR successfully corrected 96.12% of bit-flips affecting the CNN's weights, demonstrating consistent effectiveness across all layers. Figure 3b illustrates the impact of injected faults on intermediate data, i.e., the data present at the output of each CNN layer. Unlike parameters, which are only located in CONV and FC layers, intermediate data appears after every stage, including the input, ReLU activations, MaxPooling, and Flatten operations. Consequently, the number of potential faults is significantly higher. For this reason, only $2,000$ test images were used in this experiment, resulting in 7.42 million injected faults under the same conditions as before (10% margin of error and 90% confidence level). Injecting faults into intermediate data is also more complex. It requires generating ONNX files for partial models. For example, to inject a fault at layer Conv-7, the SFI4NN tool generates a model covering layers 0 to 7, simulates a bit-flip at the model's output, and then reconstructs the remaining network from layer 8 to the final output. This method is slower than parameter fault injection and could be improved in future work. Regarding the results obtained for intermediate data layers, a greater variation in sensitivity is observed across layers. Overall, intermediate data appears less sensitive than parameters, with error rates ranging from 0.05% to 0.5% for the input data layer. The effectiveness of the VANDOR protection method is illustrated with two color codes: light yellow represents corrected errors, while dark yellow corresponds to uncorrected errors. Overall, VANDOR successfully corrected 88.68% of the injected faults in intermediate data. By combining the results from the parameter and intermediate data analyses, we can assess the bit-level sensitivity across the entire CNN, as shown in Figure 3c. As expected, higher-order bits exhibit the highest sensitivity and have the greatest impact on model behavior. The VANDOR protection method achieves an overall fault correction rate of 94.90%. This result closely aligns with the performance observed for parameter protection alone, which is unsurprising given that the network contains significantly more parameters (28.5 million) than intermediate data values ($289,994$). The MSB, which determines the sign in two's complement representation, was identified as particularly sensitive, accounting for nearly 20% of the Errors/Injected faults ratio. Finally, by averaging the sensitivity rates across all CNN bits, the overall sensitivity matches with the individual observations for both parameters and intermediate data.

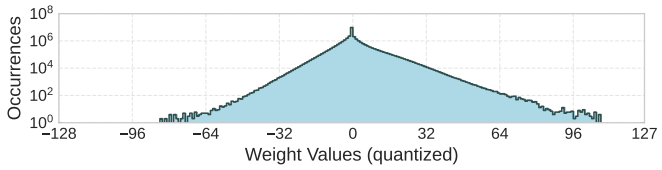
C. Impact of Pruning on Fault Sensitivity

This second study aims to examine the impact of pruning on a CNN's resilience. For this purpose, the same quantized AlexNet architecture was used. However, weight pruning was applied during the training phase. Several models were generated by varying the pruning rate from 0% to 60%, as detailed in Table III. These different pruning rates resulted in a varying proportion of zero-valued weights, ranging from 6.66% in the unpruned model to 97.44% in the most heavily pruned model. While pruning improves energy efficiency and computational speed, it inevitably affects model accuracy. In this study, the best-performing model achieved an accuracy of 84.68%, whereas the model with the highest pruning rate reached only 74.33%. Since the number of parameters per layer changes with pruning, the number of injected faults during the SFI process also varies. Consequently, for a 10% error margin, a 90% confidence level, and $10,000$ test images, the SFI4NN tool generated 14.69 million faults for the unpruned model and 13.80 million faults for the most heavily pruned model.

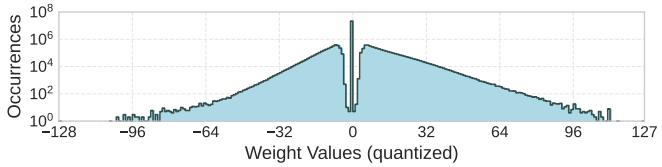
Regarding the training process during which pruning was applied, models were trained for 100 epochs using the Adam optimizer. Uniform $L1$ pruning was applied to the CONV and FC layers at epochs 20, 40, 60, and 80. Performing a fine-tuning step up to the last epochs afterward is recommended to improve model performance. The effect of pruning on the generated CNN models is illustrated in Figure 4, which shows histograms of the weight distributions across all layers of

TABLE III: Characteristics of the pruning rate study.

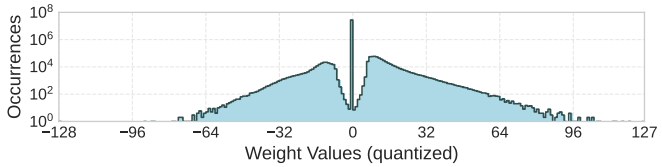
Pruning rate (%)	0	5	10	20	30	40	45	50	55	60
Zero-weight (%)	6.66	32.17	35.88	64.46	75.99	87.04	90.85	93.75	95.90	97.44
Accuracy (%)	84.05	84.68	84.47	84.20	84.05	82.56	82.13	80.67	76.92	74.33
Injected faults (million)	14.69	14.67	14.65	14.62	14.56	14.42	14.33	14.17	14.02	13.80



(a) Prune rate: 0%, zero-weight: 6.66%, accuracy: 84.05%



(b) Prune rate: 30%, zero-weight: 75.99%, accuracy: 84.05%



(c) Prune rate: 60%, zero-weight: 97.44%, accuracy: 74.33%

Fig. 4: AlexNet weight histograms with varying pruning rates.

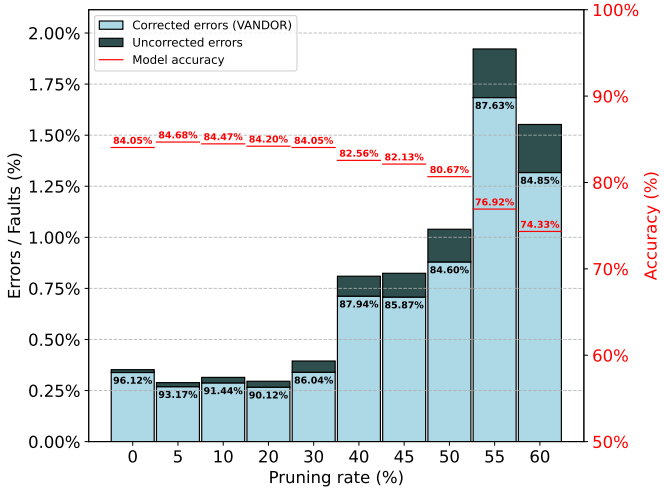


Fig. 5: CNN sensitivity based on pruning rate.

the AlexNet architecture for models with pruning rates of 0%, 30%, and 60%. Note that the Y-axis is displayed on a logarithmic scale. For the baseline model (0% pruning), most weights are concentrated around zero. This observation likely explains why the VANDOR protection method is effective, as the CNN is accustomed to operating with values close to zero. Consequently, if a fault perturbs a weight by shifting it away from zero, it is more easily identified as an outlier. For the heavily pruned models, a noticeable reduction in near-zero values is observed. This decrease aligns with the pruning process, which removes less significant connections. However, it's important to highlight that in the ONNX file generated by Brevitas, pruned connections reappear as zero-valued weights. This explains their significant presence in the last two histograms.

Figure 5 presents the sensitivity results of AlexNet CNN models at different pruning rates. The X-axis shows the pruning rates, ranging from 0% to 60%, while the left Y-axis represents the Error/Fault ratio and the right Y-axis indicates model accuracy. The figure reveals that increasing the pruning rate makes the model more sensitive to faults. Sensitivity rises from approximately 0.3% in models with low pruning rates to nearly 2% in heavily pruned CNNs. However, models achieving over 84% accuracy exhibit stable sensitivity levels, with pruning rates between 0% and 30% maintaining a sensitivity below 0.5%. This result suggests that a quantized CNN can be pruned while retaining both good prediction performance and strong resilience against SEUs. Regarding the VANDOR protection method, its effectiveness gradually decreases as pruning increases. VANDOR corrects up to 96.12% of faults in an unpruned model, but this rate drops to 84.60% in models with higher pruning rates. Despite this decline, the results remain encouraging, highlighting the importance of carefully balancing pruning strategies and protection methods to achieve an optimal trade-off between model accuracy and inferences speed, fault tolerance and hardware cost.

V. Conclusion

This article presents SFI4NN, a fault injection method based on a statistical approach designed to assess resilience of quantized and pruned CNNs. The proposed method was validated on a quantized AlexNet with CIFAR-10 as a case study. Additionally, we evaluated the effectiveness of the VANDOR hardware protection solution in this context, specifically examining its performance in pruned models. While VANDOR offers strong protection, its effectiveness drops with higher pruning rates. This study highlights the trade-offs between pruning and fault tolerance, emphasizing the need for a balanced strategy considering accuracy, inference speed, fault resilience, and hardware cost.

Acknowledgment

This work was supported by the French Directorate General of Armaments (DGA), with funding from the Defense Innovation Agency (AID) and Inria.

References

[1] Alessandro P. *Xilinx/brevitas*. 2023. DOI: 10.5281/zenodo.3333552. URL: https://doi.org/10.5281/zenodo.3333552.

[2] M. Paganini and J. Forde. "Streamlining tensor and network pruning in pytorch". In: *arXiv preprint arXiv:2004.13770* (2020).

[3] Annachiara Ruospo et al. "Assessing convolutional neural networks reliability through statistical fault injections". In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2023, pp. 1–6.

[4] F Libano et al. "How reduced data precision and degree of parallelism impact the reliability of convolutional neural networks on FPGAs". In: *IEEE Transactions on Nuclear Science* 68.5 (2021), pp. 865–872.

[5] Zhen Gao et al. "Reliability evaluation of FPGA based pruned neural networks". In: *Microelectronics Reliability* 130 (2022), p. 114498.

[6] Ioanna Souvatzoglou et al. "Assessing the reliability of FPGA-Based Quantised Neural Networks under Neutron Irradiation". In: *IEEE Transactions on Nuclear Science* (2024).

[7] Wilfread Guillemé et al. "VANDOR: Mitigating SEUs into Quantized Neural Networks". In: *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE. 2024, pp. 1–6.

[8] Alberto Bosio et al. "A reliability analysis of a deep neural network". In: *2019 IEEE Latin American Test Symposium (LATS)*. IEEE. 2019, pp. 1–6.

[9] Yu-Hsin Chen et al. "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9.2 (2019), pp. 292–308.

[10] Guanpeng Li et al. "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.

[11] Marcello Traiola, Angeliki Kritikakou, and Olivier Sentieys. "harDNNing: a machine-learning-based framework for fault tolerance assessment and protection of DNNs". In: *2023 IEEE European Test Symposium (ETS)*. IEEE. 2023, pp. 1–6.

[12] Yangchao Zhang et al. "Estimating vulnerability of all model parameters in DNN with a small number of fault injections". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2022, pp. 60–63.

[13] Elbruz Ozen and Alex Orailoglu. "Just say zero: Containing critical bit-error propagation in deep neural networks with anomalous feature suppression". In: *Proceedings of the 39th International Conference on Computer-Aided Design*. 2020, pp. 1–9.

[14] Wenda Wei et al. "An Approximate Fault-Tolerance Design for a Convolutional Neural Network Accelerator". In: *IT Professional* 25.4 (2023), pp. 85–90.

[15] Stéphane Burel, Adrian Evans, and Lorena Anghel. "Mozart+: Masking outputs with zeros for improved architectural robustness and testing of dnn accelerators". In: *IEEE Transactions on Device and Materials Reliability* 22.2 (2022), pp. 120–128.

[16] Le-Ha Hoang, Muhammad Abdullah Hanif, and Muhammad Shafique. "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2020, pp. 1241–1246.

[17] Régis Leveugle et al. "Statistical fault injection: Quantified error and confidence". In: *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE. 2009, pp. 502–506.

[18] ONNX Runtime developers. *ONNX Runtime*. https://onnxruntime.ai/. Version: x.y.z. 2021.

[19] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[21] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[22] Junjie Bai, Fang Lu, Ke Zhang, et al. *Onnx: Open neural network exchange*. 2019.

[23] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).