HTAG-eNN: Hardening Technique with AND Gates for Embedded Neural Networks

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

HTAG-eNN: Hardening Technique with AND Gates for Embedded Neural Networks

Wilfread Guillemé
wilfread.guilleme@inria.fr
IRISA/INRIA, Univ. Rennes
Lannion, France

Angeliki Kritikakou
angeliki.kritikakou@irisa.fr
IRISA/INRIA, IUF, Univ. Rennes
Rennes, France

Youri Helen
youri.helen@intradef.gouv.fr
DGA MI
Bruz, France

Cédric Killian
cedric.killian@univ-st-etienne.fr
Lab. Hubert Curien, CNRS,
Univ. Jean Monnet
Saint-Étienne, France

Daniel Chillet
daniel.chillet@irisa.fr
IRISA/INRIA, Univ. Rennes
Lannion, France

ABSTRACT

Embedded Neural Networks (NNs) face significant challenges due to Single-Event Upsets (SEUs), compromising their reliability. To address this challenge, previous works study SEU layers sensitivity of AI models. Contrary to these techniques, remaining at high level, we propose a more accurate analysis, highlighting that, except for the last layer, faults transitioning from 0 to 1 significantly impact classification outcomes. Based on this specific behavior, we propose a simple hardware block able to detect and mitigate the SEU impact. Obtained results show that HTAG protection efficiency is near 96.85% for the LeNet-5 CNN inference model, suitable for an embedded system. This result can be improved with other protection methods for the classification layer. Additionally, it significantly reduces area overhead and critical path compared to existing approaches.

CCS CONCEPTS

• **Hardware → Redundancy**.

KEYWORDS

Fault Tolerance, Optimization, Protection, Redundancy, Robustness

1 INTRODUCTION

A wide range of application domains uses artificial intelligence algorithms (based on Neural Network - NN) to solve several tasks, such as image and speech recognition, natural language processing and autonomous control systems. When they are used in embedded systems for safety-critical domains, reliability becomes of paramount importance. As a result, it is necessary to protect these systems from reliability threats, especially from transient faults occurring due to environmental conditions, e.g., high temperature and high-energy electromagnetic radiation, which are considered one of the most important faults sources [1]. Such faults, temporarily inserted to the hardware components, flip the value stored in flip-flops leading to Single-Event Upsets (SEUs). It can propagate during system execution and finally impact the NN output, modifying the final inference decision.

As the size of NNs has significantly increased in recent years, massive flows of data occur during CNN inference. This evolution has led to the design of dedicated hardware architectures tailored for NN-based embedded systems, where execution occurs layer by layer. For instance, EyerissV2 [2] is an accelerator architecture designed for NNs and its design employs a Network-on-Chip (NoC) [3] to efficiently handle weights, biases, and data transfers. In these architectures, not only the NN parameters, such as weights and biases, need to be locally stored, but also the intermediate data, which require temporary storage when passed to the next layer. While majority of existing approaches consider SEUs occurring only to the parameters [4] [5], our analysis provides a study on intermediate data and also explores the fault impact considering the direction of bit change. This reveals that SEU from 0 to 1 have a much greater effect on the classification result than 1 to 0 SEU. Other studies have also highlighted an asymmetry in the fault direction impact for the Floating-Point (FP) data format [6].

Regarding fault protection of these algorithms, typical methods are generally based on hardware or information redundancy on sensitive flip-flops, such as Triple Modular Redundancy (TMR) [7], Dual Modular Redundancy (DMR) [8] and Error Correcting Code (ECC) using Hamming technique [9]. Based on our accurate level analysis of faults impact, we propose a selective fault tolerant technique that forces to zero faulty values instead of correcting them. This light-weight solution, named Hardening Technique with AND Gates (HTAG), is based on duplicating sensitive flip-flops and inserting AND gates to drive the hardware design. To validate our proposal in the context of embedded systems, we consider a LeNet-5 [10] model with different FP formats. The proposed approach is driven by a comprehensive analysis of the fault impact on weights, biases and intermediate data, considering the criticality of layers, data-types and the SEU direction. From the obtained use-case results, we show that HTAG achieves a protection efficiency of 96.85% for the most fault-tolerant model using the float6 data format, with few hardware and critical path overhead against others compared methods.

The rest of this paper is organised as follows: Section 2 presents the state-of-the-art regarding NN fault analysis and protection. Section 3 provides a comprehensive study of SEU impact on the NN classification. Section 4 presents the proposed Hardware Technique with AND Gates implemented as voter block. Comparisons with others methods are also presented in this section, and Section 5 concludes this paper and presents some perspectives.

2 RELATED WORK

Several existing works evaluate the reliability of NN systems through fault analysis and protection. Regarding faults analysis, such paper focus on NN parameters, especially weights [4] [5]. Considering systems that execute NNs layer by layer, intermediate data needs to be stored temporarily and is therefore also sensitive to SEUs. The analysis framework for deep learning accelerators, presented in [11], enables to extract possible datapath and some flip-flops containing weights, biases and data between layers. Authors in [12] provide a study for different NNs and different data formats, i.e. fixed-point and FP. Other approaches explore the criticality of the bits under SEU on IEEE754 values for CNN weights [13]. They show that FP format has fewer sensitive bits but error may be greater since most significant bits have exponential impact on coded value. ARES [14] framework explores the fault susceptibility among models, layers, and activations, considering faults in the memory. Since fault injection simulation can be very time-consuming, statistical fault injection methods are proposed by [15]. Demonstrating that with around 1% of possible injected faults, they can achieve a CNN reliability estimate close to the exhaustive result. These previous studies clearly show that the SEU sensitivity is different depending on the fault position within the NN architecture. However, although some have noticed that there is an asymmetry depending on the SEU direction [6], there is no study provided to quantify how problematic SEU from 0 to 1 are.

Regarding fault protection, typical approaches are based on hardware or information redundancy at different levels. Architecture-level resilience can be achieved by replication of hidden neurons and their associated connections [16]. Furthermore, selective replication is possible by detecting the more sensitive elements and duplicating the inputs, biases, weights [17] and critical neurons [18]. ApFT [19] addresses fault tolerance in CNNs with a systolic array dataflow architecture. It accomplishes this by duplicating convolution filters while taking into account the specific architectural characteristics. Micro-architecture level resilience is based on data precision adjustment, (e.g., storing weights with different formats [20]), selective bit hardening (e.g. harden the MSB considering fixed-point arithmetic [12]) and duplication of computing units (e.g., replication only for the cells that affect the higher bit output of a ripple-carry adder [21]). Other approaches improve resilience of AI models by mitigating the impact of hardware failures on CNN operations. For instance, FT-ClipAct [22] constraints excessively high weight values through a threshold mechanism for faults affecting the memory storing network weights. MOZART+ [23] is applied over the output stationary dataflow within systolic arrays, where a multiplexer is used to systematically turn off processing elements that are susceptible to faults.

Based on findings from other studies, we performed an additional study for different FP formats considering data transferred between layers. Our analysis also explores the SEU impact considering the direction of bit change and reveals that SEUs transitioning from 0 to 1 have a more significant effect on the classification outcome than 1 to 0 SEU. This information is used to drive the design of the proposed fault protection technique, which is based on selectively duplicating flip-flops and inserting logical AND gates to mitigate the SEU impact with a very low hardware overhead.

Layer type		# Data	Output shape	# Parameters
Image	Input	784	[1, 28, 28]	0
CONV-1 (Convolution)	Conv-1	4,704	[6, 28, 28]	156
	Sig-2	4,704	[6, 28, 28]	0
	Pool-3	1,176	[6, 14, 14]	0
CONV-2	Conv-4	1,600	[16, 10, 10]	2,416
	Sig-5	1,600	[16, 10, 10]	0
	Pool-6	400	[16, 5, 5]	0
FC-1 (Fully Connected)	Flat-7	400	[400]	0
	Lin-8	120	[120]	48,120
	Sig-9	120	[120]	0
FC-2	Lin-10	84	[84]	10,164
	Sig-11	84	[84]	0
FC-3	Lin-12	10	[10]	850
	Soft-13	10	[10]	0
Output	**Total**	**15,012**		**61,706**

Table 1: Details of the considered LeNet-5 [10] architecture.

3 SEU-SENSITIVITY ANALYSIS

In this section, we present the analysis conduct on simple classification LeNet-5 [10], based on convolution, pooling, linear and activation layers. Same conclusions could be obtained for more bigger CNNs.

3.1 Use-case Neural Network architecture

Our use-case is a LeNet-5 [10] model trained with the MNIST database and implemented with the PyTorch library. Table 1 describes its characteristics. The targeted model consists of two convolutional layers followed by three fully connected layers. NN parameters, determined during the training phase, are only on these layers. After each layer, an activation function is applied, such as sigmoid. Average pooling layers are also used to reduce the data shape after each convolutional layer. The last layer implements a softmax function for classification, assigning a probability to each class. The class with the highest probability represents the NN's classification outcome.

Considering that the NN is executed over an embedded system, we adapt the model parameters and data to a reduced FP format. More precisely, we conduct simulations using various custom data formats, i.e., float4 (1, 2, 1), float6 (1, 3, 2), float8 (1, 4, 3), float12 (1, 4, 7), and float16 (1, 5, 10). The notation (S, E, M) corresponds to S for the sign, E for the exponent part and M for the mantissa part. E.g., the float8(1, 4, 3) format corresponds to 1 bit for the sign, 4 bits for the exponent part, and 3 bits for the mantissa part.

3.2 Fault injection framework

We consider SEUs occurring during inference and we inject faults on parameters, i.e., weights and biases, as well as on the intermediate data transferred between layers. Note that, for parameters, the fault injection campaign is limited to the convolutional and fully connected layers, as these are the layers that exclusively contain these parameters. Conversely, intermediate data can be selected on all layers. As soon as the fault is injected, the model is executed with these new parameters by updating the Python dictionary object that maps each layer to its parameters. However, injecting faults into the data is slightly different because the data within the NN

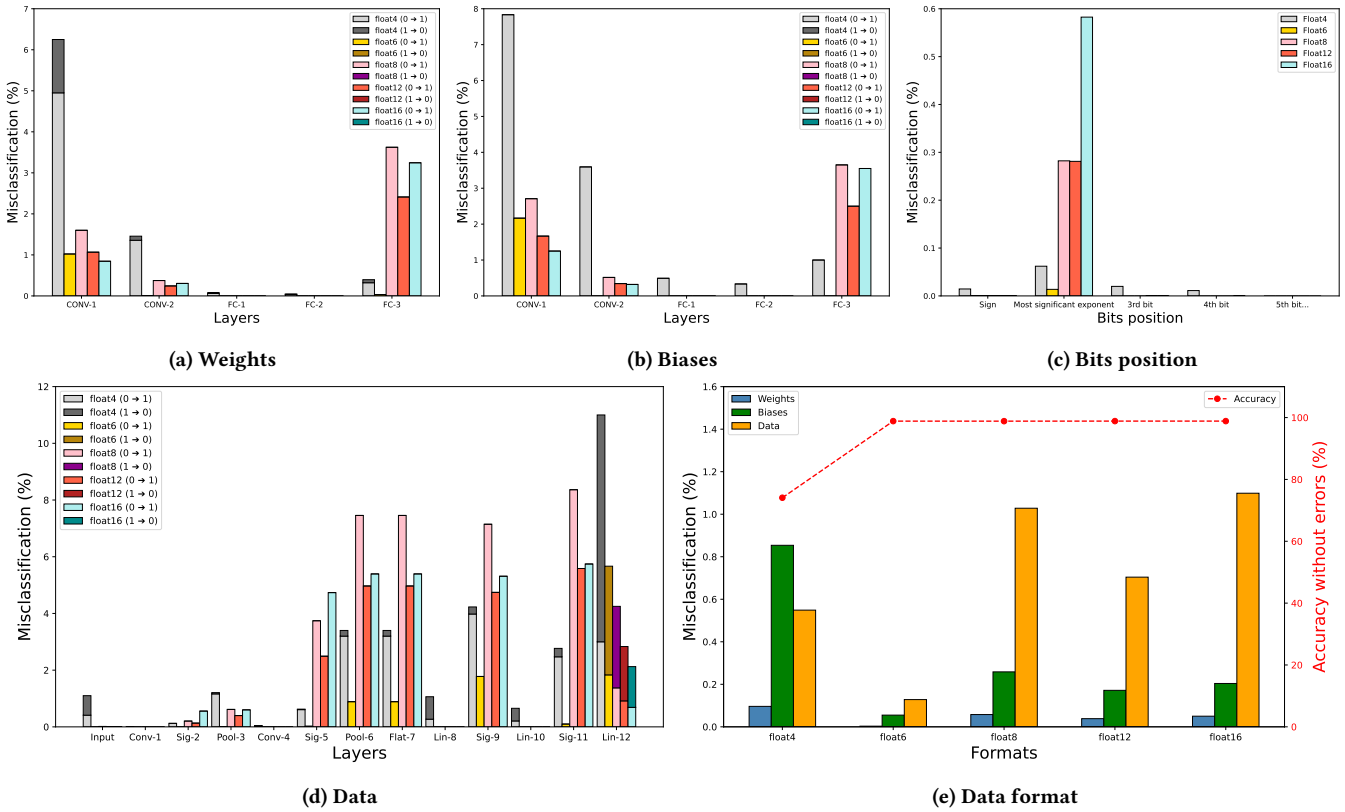


Figure 1: SEU injection fault campaign results: Impact of SEU direction per layer on a) Weights, b) Biases, d) Data ; c) Impact of SEU on bit position ; and e) SEU sensitivity and Neural Network accuracy with respect to data format.

are not directly accessible by default through PyTorch. In this case, the NN is executed layer by layer to retrieve and modify the data targeted for the fault injection. The data before entering the NN, i.e. the grayscale image representing a number, is also subjected to the fault injection campaign.

The SEU injection campaign is exhaustive, and thus, it explores all potentially faulty bits possibilities. The obtained results are compared to the golden reference, produced by the fault-free inference of the NN. If there is a classification difference between both executions, the faulty bit is considered as sensitive and is tagged as a bit to be protected.

3.3 Sensitivity per layer and data type

Obtained results of the fault injection campaign indicate that layers of the studied NN do not have the same sensitivity to SEUs. For weights, as presented in Figure 1a, it appears that sensitive layers are the convolutional layers located at the network's input and at the last linear layer. When a fault is inserted into the weight of a convolutional filter, the fault impacts the whole image. Consequently, a fault at the weight level can have a significant impact on the network's accuracy. Furthermore, we can note that, the last linear layer is also sensitive because it is close to the final classification decision.

Similar results are observed for biases in Figure 1b. However, biases are few compared to weights, since there is only one bias per channel for the convolutional layers and one bias for all the neurons of linear layers. It is interesting to highlight that sensitive layers are also the ones with the fewest parameters.

Figure 1d illustrates results for data transferred between layers. We observe that layers exhibit varying sensitivities to SEUs over the transferred data, and in particular for activation layers. For example, sigmoid function, which produces values between 0 and 1, a fault in high-weight bit can lead to a significant increase in the output value, potentially compromising subsequent layers. Furthermore, we observe that fault sensitivity increases the closer the fault is to the NN output layer. This is due to the architectural model design, which progressively reduces the volume of data transferred between layers to finally produce a set of 10 values corresponding to the defined MNIST classes.

Figure 1c presents results regarding impact of the faulty bit position to misclassification. This figure confirms that sign and exponent bits mostly impact NN accuracy. The obtained results are coherent with similar studies such as [4] [12]. Sign and the highest-weighted exponent bits exhibit high sensitivity for all formats except the float4. Therefore, we can achieve high level protection, while reducing the hardware overhead, by considering only these two types of bits for protection, rather than all bits in the FP format.

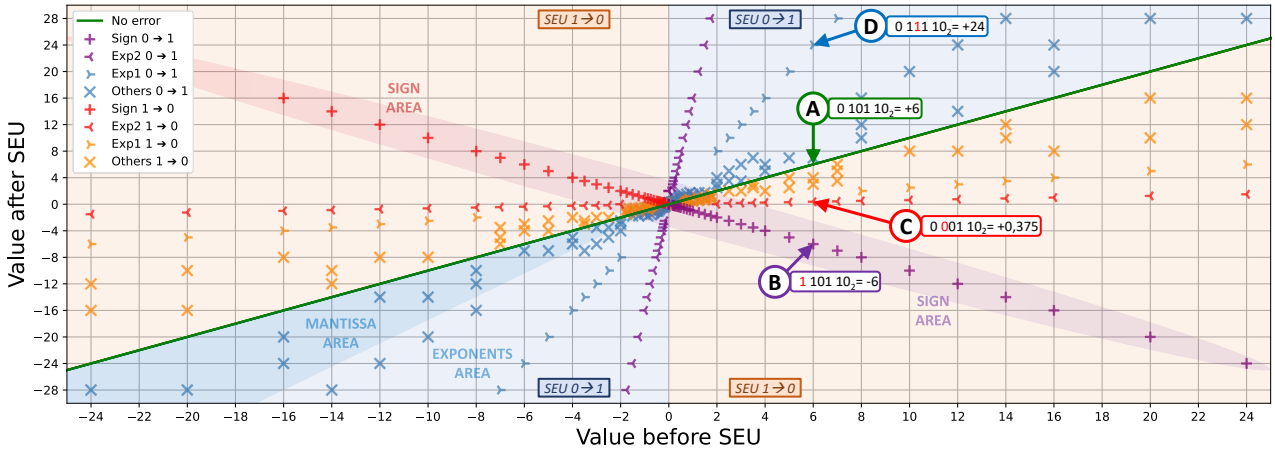


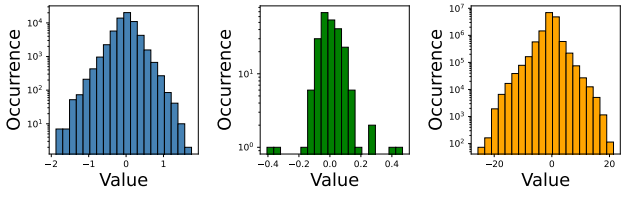
Figure 2: SEUs impact on the float6 (1, 3, 2) format.

Figure 1e combines results of all faults injection campaigns, i.e., weights, biases and intermediate data for all FP formats. From this figure, we can conclude that data flowing between network layers are generally more susceptible to produce misclassification. This observation can be explained as follows: let's consider a fully connected layer. When a fault is introduced on a weight, its impact is confined to a single connection. Regarding bias, the fault is partially mitigated by the subsequent activation function. However, for data transferred between two layers, a fault has an effect on all connections involving this data. This is true, except for the neural network using the 4-bit FP format, but it fails to achieve a high level classification rate.

Overall, Figure 1 shows that the float6 (1, 3, 2) format is the most tolerant to faults. Note that, this floating point format is well sized for such NN. On the one hand, smaller formats, such as float4, do not contain enough bits to ensure sufficient encoded values accuracy. On the other hand, float8, float12 and float16 formats contain too many insignificant bits that induce higher hardware implementation cost.

3.4 Impact of the SEU direction

This section presents the fault direction impact on the NN accuracy, i.e., flipping a bit from 0 to 1 or from 1 to 0. Figure 1 also highlights the SEU direction impact. Results of the fault injection campaign show that SEUs from 1 to 0 cause very few NN misclassifications. To better understand this observation, firstly, we recover all possible values when executing the fault-free model, and then, we inject faults on all these values. Figure 3 presents all possible values during the fault-free NN inference. The histogram 3a is referring to weights and it shows that typical values range of approximately +/−2. The histogram 3b corresponds to biases and their values tend to fluctuate within a narrower range of approximately +/−0.5. However, data, whose values are depicted in histogram 3c, traverse through different network layers, and thus, values can span from +/−24. Overall, histograms show that NN operates with values centered near 0.



(a) Weights **(b) Biases** **(c) Data**

Figure 3: Histograms for a) Weights, b) Biases, c) Data.

Figure 2 presents the impact of the SEU on the range of values for weights, bias and data considering the float6 (1, 3, 2) format. As we can observe from this figure, bits toggle from 0 to 1 often lead to important deviations from the original values represented in the blue areas of the figure. The same observation holds for faults impacting the exponent bits. The value impacted by a fault is consistently moved further compared to faults impacting the mantissa bits. This is a logical result given that the mantissa is less significant, as illustrated by the lower left part of the figure. Furthermore, the histogram analyses, regarding the values of weights, bias and data, have shown that NN mainly operates with values close to zero. Therefore, when SEU change a value from 1 to 0, it tends to bring the value closer to zero, excluding the sign bit. This observation clarifies why SEUs from 1 to 0 do not have a significant impact on the NN classification. To further analyse the impact of SEUs, we consider the following example: let consider the value 6, expressed as 0 101 10_2 in float6 (1, 3, 2) format, denoted as point A in Figure 2. When a bit inversion occurs on the initial sign bit, the value immediately changes to -6, marked as point B. An inversion on the most significant exponent bit, results in the value shifting towards zero, i.e. 0.375 represented by point C. Lastly, a change from 0 to 1 in a bit causes a sudden surge in the value to 24 (point D). Upon close examination of this figure, it becomes clear that if a fault affects a bit, it is preferable to considered this bit as zero to

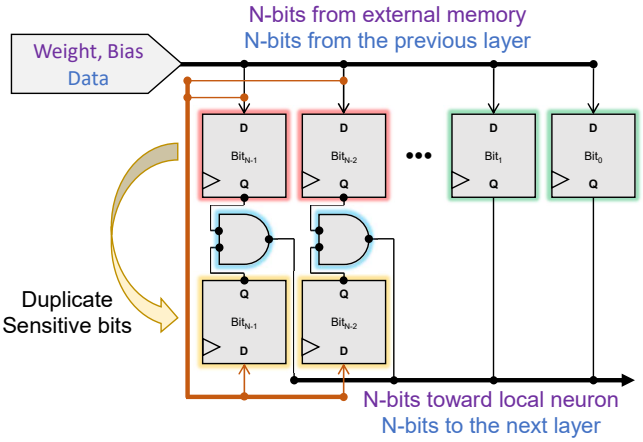


Figure 4: HTAG hardware implementation.

reduce the fault impact. This aligns with the typical NN working range and has a lesser disruptive impact on the network.

4 HARDWARE PROTECTION

In this section, Hardening Technique with AND Gate (HTAG) is presented. This technique is driven by the analysis presented in Section 3, hence taking into account layer sensitivity and the bits, depending on the data type. The main idea of the proposed technique is to duplicate all flip-flops storing a sensitive bit and add an AND logic gate that acts as a voter block. Based on the truth table, the voter results in favor of the zero logic state when the two input bits are different. If a bit equal to 0 is impacted by a fault, i.e. it is flipped to 1, the vote will result to a 0 and the fault impact is corrected. Conversely, if a bit equal to 1 is impacted by fault, i.e. it is flipped to 0, the vote will produce a 0. Leading to maintain the fault, which has generally no impact on the classification result as presented on the section 3.

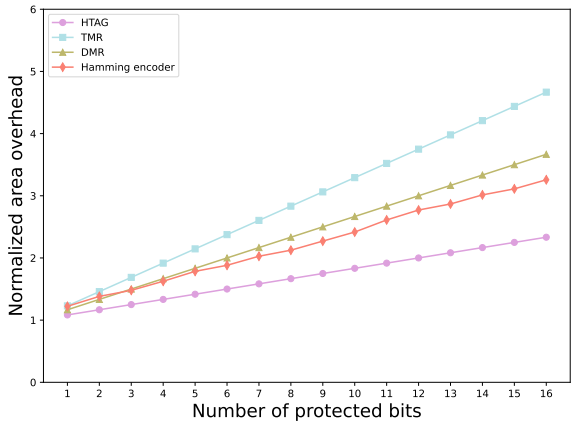
Figure 4 shows the HTAG hardware implementation. As highlighted during the study regarding the impact of the SEUs direction in subsection 3.4, replacing a 1 by a 0 does not have severe consequences for NN classification. This strategy is simple to implement, requiring only minimal additional hardware resources, and it increases the critical path by just one combinational AND gate.

We further evaluate the proposed HTAG method regarding: i) the effectiveness of protection, ii) the area overhead, and iii) the critical path compared to common protection techniques, such as TMR, DMR, and Hamming code. To provide a fair analysis, we assume that all techniques have the same number of bits to protect.

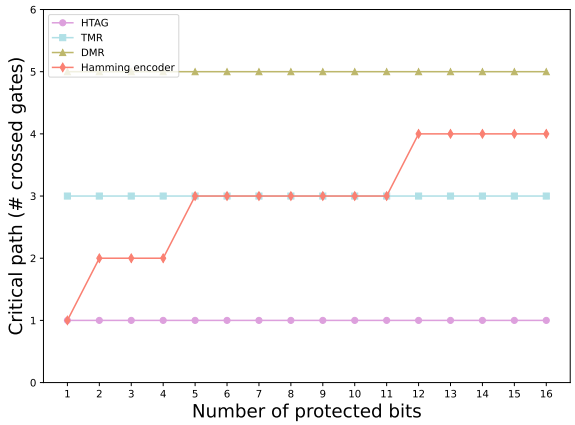
Regarding the effectiveness of protection, TMR, DMR, and Hamming code achieve a 100% success rate in case of SEU error, as they are able to correct one faulty bit. However, if the target application can tolerate some errors or has limited resources, HTAG is a promising solution as it significantly reduces the hardware overhead. The fault injection campaigns revealed a loss of only 3.15% for the most fault-resistant model, based on the float6 format, as indicated in Table 2. Furthermore, the analysis of Figure 1d shows that the HTAG technique is less effective for the last layer, where the majority of misclassifications is due to SEUs transitioning from 1 to 0. This

Format	# Faults	# Errors	# 0 ⇒ 1	# 1 ⇒ 0	HTAG
Float4	3,057,440	6,159	5,559	600	90.25%
Float6	4,586,160	1,364	1,321	43	96.85%
Float8	6,114,880	15,984	15,931	53	99.67%
Float12	9,172,320	16,328	16,275	53	99.68%
Float16	12,229,860	32,915	32,835	80	99.76%

Table 2: Performance analysis of HTAG with the previous SEU fault injection campaign.



(a) Area overhead



(b) Critical path estimation

Figure 5: HTAG comparison against other hardening methods in a) Area overhead, b) Critical path estimation.

result is explained since data at the end of the last layer and before the softmax function are negative except for the predicted class (Not visible on histogram 3c). There are therefore many sign bits at the logic state 1 with a fault that can only go towards logic state 0. This affects the HTAG's protection performance. Therefore, the combination of HTAG with another protection method applied at the final layer is a promising solution.

HTAG stands out for its efficiency in terms of hardware overhead required for protection. Indeed, HTAG only needs a single

additional logic gate and a single flip-flop for bit duplication, which is significantly less compared to others protection techniques, as illustrated in Figure 5a. While TMR is the most effective technique for bit correction, it is also the most costly one. Regarding DMR, although it reduces the flip-flop count compared to triplication, the critical path is heavily impacted. Finally, Hamming code has high hardware overhead when the number of bits to protect is very low, while it becomes attractive only as the number of bits to protect increases. As shown by the sensitivity analysis, the number of sensitive bits to protect is small, and thus, this method is not appropriate for this case due to its high hardware overhead. Note that, the presented results regarding the overhead of Hamming code consider only the part related to the encoder, leading to an lower bound estimation regarding the hardware overhead.

Another significant advantage of HTAG is its low impact in terms of critical path time. Figure 5b presents an estimation of the impact on critical path due to hardware implementations of the three methods by counting the number of crossed logic gates. Unlike TMR's voting block requiring three logic gates, HTAG only needs a single simple logic gate to mitigate the fault. Regarding DMR, although it reduces the flip-flop count compared to triplication technique, its critical path time is heavily impacted due to the feedback loop in the voting block. As for Hamming code, its critical path time gradually increases with the number of bits to protect, as more XOR gates are required to compute the parity bits.

5 CONCLUSION AND DISCUSSION

In this paper, we initially conducted a comprehensive analysis of SEU affecting a LeNet-5 model that can be used in embedded systems. We observed that there is a sensitivity difference among different layers of this CNN. Convolutional layers and the output layer are identified as the most sensitive ones to faults occurring to NN parameters. Concerning the intermediate data, as faults occur closer to the output, they become more sensitive. By studying SEUs direction, we observed that the NN is more tolerant for faults leading to bit flips from 1 to 0. Using this observation, we proposed HTAG, which duplicates sensitives bits and adds a single AND logic gate for protection. HTAG provides robust hardening performance with negligible hardware overhead and minimal increase of the critical path compared to existing methods. It brings erroneous data closer to values near zero, which are typical values for NN. Obtained results show that HTAG achieves a protection efficiency of 96.85% for the most fault-tolerant model using the float6 data format, with low hardware and critical path overhead. Taking into account the last layer which is not suitable for the HTAG protection.

Our fault injection analysis is limited by the required simulation time. To address this limitation, as future work we will consider to introduce faults exclusively in the most significant bits, as it has been shown that the remaining bits are less sensitive. Furthermore, to deal with larger NN architectures, statistical fault injection could be used instead of exhaustive fault injection. As future work, we could explore the applicability of the HTAG method to bigger NNs and assessing its protection effectiveness against Multiple Bit Upsets (MBUs) and multiple Single-Event Upsets (SEUs) occurring on different layers, a possible scenario when a system executes the NN layer by layer.

REFERENCES

[1] Semeen Rehman, Muhammad Shafique, and Jörg Henkel. *Reliable software for unreliable hardware: A cross layer perspective*. Jan. 2016, pp. 1–192. ISBN: 978-3-319-25770-9. DOI: 10.1007/978-3-319-25772-3.

[2] 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.

[3] Shashi Kumar et al. "A network on chip architecture and design methodology". In: *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI 2002*. IEEE. 2002, pp. 117–124.

[4] Annachiara Ruospo et al. "Evaluating Convolutional Neural Networks Reliability depending on their Data Representation". In: *Euromicro Conference on Digital System Design (DSD)*. 2020, pp. 672–679. DOI: 10.1109/DSD51259.2020.00109.

[5] 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.

[6] 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.

[7] Fabiano Libano et al. "Selective hardening for neural networks in FPGAs". In: *IEEE Transactions on Nuclear Science* 66.1 (2018), pp. 216–222.

[8] Ramin Rajaei. "Design of a Radiation Hardened Register File for Highly Reliable Microprocessors". In: *International Journal of Engineering and Manufacturing (IJEM)* 6 (Feb. 2016). DOI: 10.5815/ijem.2016.05.02.

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

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

[11] Yi He, Prasanna Balaprakash, and Yanjing Li. "Fidelity: Efficient resilience analysis framework for deep learning accelerators". In: *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2020, pp. 270–281.

[12] 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*. SC '17. Denver, Colorado: Association for Computing Machinery, 2017. ISBN: 9781450351140. DOI: 10.1145/3126908.3126964. URL: https://doi.org/10.1145/3126908.3126964.

[13] Mohamed A. Neggaz et al. "Are CNNs Reliable Enough for Critical Applications? An Exploratory Study". In: *IEEE Design & Test* 37.2 (2020), pp. 76–83. DOI: 10.1109/MDAT.2019.2952336.

[14] Brandon Reagen et al. "Ares: A framework for quantifying the resilience of deep neural networks". In: *Proceedings of the 55th Annual Design Automation Conference*. 2018, pp. 1–6.

[15] A 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.

[16] Schuyler Eldridge and Ajay Joshi. "Exploiting hidden layer modular redundancy for fault-tolerance in neural network accelerators". In: *Proc. Boston area ARChitecture (BARC) Workshop*. 2015.

[17] Fernando Morgado Dias and Ana Antunes. "Fault Tolerance improvement through architecture change in Artificial Neural Networks". In: *International Symposium on Intelligence Computation and Applications*. Springer. 2008, pp. 248–257.

[18] Iljoo Baek et al. "FT-DeepNets: Fault-Tolerant Convolutional Neural Networks with Kernel-based Duplication". In: *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2022, pp. 1878–1887. DOI: 10.1109/WACV51458.2022.00194.

[19] Wenda Wei et al. "An Approximate Fault-Tolerance Design for a Convolutional Neural Network Accelerator". In: *IT Professional* 25.4 (2023), pp. 85–90. DOI: 10.1109/MITP.2023.3264849.

[20] Sung Kim et al. *MATIC: Learning Around Errors for Efficient Low-Voltage Neural Network Accelerators*. 2018. arXiv: 1706.04332 [cs.NE].

[21] Hamid Reza Mahdiani, Sied Mehdi Fakhraie, and Caro Lucas. "Relaxed fault-tolerant hardware implementation of neural networks in the presence of multiple transient errors". In: *IEEE transactions on neural networks and learning systems* 23.8 (2012), pp. 1215–1228.

[22] 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)*. 2020, pp. 1241–1246. DOI: 10.23919/DATE48585.2020.9116571.

[23] Stephane 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.