HAL
open science

harDNNing: a machine-learning-based framework for fault tolerance assessment and protection of DNNs

Marcello Traiola, Angeliki Kritikakou, Olivier Sentieys

To cite this version:

Marcello Traiola, Angeliki Kritikakou, Olivier Sentieys. harDNNing: a machine-learning-based framework for fault tolerance assessment and protection of DNNs. ETS 2023 - IEEE European Test Symposium, May 2023, Venise, Italy. pp.1-6. hal-04087375

HAL Id: hal-04087375

https://hal.science/hal-04087375v1

Submitted on 3 May 2023

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.

harDNNing: a machine-learning-based framework for fault tolerance assessment and protection of DNNs

Marcello Traiola, Angeliki Kritikakou, Olivier Sentieys
Inria, Univ Rennes, CNRS, IRISA
marcello.traiola@inria.fr, angeliki.kritikakou@irisa.fr, olivier.sentieys@inria.fr

***Abstract*—Deep Neural Networks (DNNs) show promising performance in several application domains, such as robotics, aerospace, smart healthcare, and autonomous driving. Nevertheless, DNN results may be incorrect, not only because of the network intrinsic inaccuracy, but also due to faults affecting the hardware. Indeed, hardware faults may impact the DNN inference process and lead to prediction failures. Therefore, ensuring the fault tolerance of DNN is crucial. However, common fault tolerance approaches are not cost-effective for DNNs protection, because of the prohibitive overheads due to the large size of DNNs and of the required memory for parameter storage. In this work, we propose a comprehensive framework to assess the fault tolerance of DNNs and cost-effectively protect them. As a first step, the proposed framework performs datatype-and-layer-based fault injection, driven by the DNN characteristics. As a second step, it uses classification-based machine learning methods in order to predict the criticality, not only of network parameters, but also of their bits. Last, dedicated Error Correction Codes (ECCs) are selectively inserted to protect the critical parameters and bits, hence protecting the DNNs with low cost. Thanks to the proposed framework, we explored and protected two Convolutional Neural Networks (CNNs), each with four different data encoding. The results show that it is possible to protect the critical network parameters with selective ECCs while saving up to 83% memory w.r.t. conventional ECC approaches.**

***Index Terms*—Reliability Analysis; Fault Tolerance; Machine Learning; Neural Networks.**

I. Introduction

Deep Neural Networks (DNNs) [2] are currently one of the most intensively and widely used predictive models in the field of Machine Learning (ML). DNNs have proven to give very good results for many complex tasks and applications, such as object recognition in images/videos, natural language processing, satellite image recognition, robotics, aerospace, smart healthcare, and autonomous driving.

The network results can be affected not only by the inaccuracy of DNNs models, but also by hardware faults. The underlying hardware, where the DNN is executed, is subject to faults due to several sources, e.g., defects undetectable at time-zero post-fabrication testing that manifest themselves later in the field of application, silicon ageing, and environmental stress, such as heat, humidity, vibration, and radiation [3]. Although DNN models have the capability to circumvent to a large extent hardware faults during the learning process, faults can still occur after training. Thus, inference can be significantly affected, leading to DNN prediction failures that are likely to lead to a detrimental effect on the application [4]–[6]. Therefore, ensuring the reliability of DNN is crucial, especially when deployed in safety- and mission-critical applications.

However, common reliability approaches, such as complete Triple Modular Redundancy (TMR) and Error Correction Codes (ECC), are not cost effective since they incur prohibitive overheads due to the large-sized DNNs and memories required for parameter storage. Therefore, to ensure the DNN reliability in a cost effective manner, the DNN resilience must be properly evaluated, the most critical parts should be identified and such an information can be used to drive the network protection.

The evaluation of the DNN resilience is typically based on simulation-based, hardware-based and radiation-based Fault Injection (FI) [7], [8]. Simulation-based fault-injection is a commonly used technique as it is a low cost and high controllable approach. Software simulation-based FI is usually performed due to reduced FI time. Several approaches exist based on different models, e.g., based on PyTorch [9] and TensorFlow [10], injecting permanent or transient faults at several fault locations, e.g., weights, activation, and operators [5]. Hardware-based FI is more time consuming, but it is more accurate as it takes into account the underlying hardware. For instance, approaches inject permanent and transient faults during inference at the hardware accelerator RTL [11]. To reduce simulation time, hybrid (or cross-layer) approaches are proposed to obtain fault models using hardware information and injected at the software level, e.g., high-level design information from architectural descriptions is used to model faults [12] and each line of code is mapped to the corresponding hardware component of the DNN accelerators [13], [14].

However, exhaustive FI is not feasible, since every neuron and synapse is a FI candidate, whereas each FI requires performing inference on the complete test set to assess the fault effect, being intractable from a computation point of view. Therefore, the majority of approaches evaluate DNN resilience through statistical FI. Following such an approach, only the criticality of the parameters where faults have been injected can be estimated. To protect the DNN in a cost effective way, the criticality of all parameters should be evaluated.

To address this limitation, recent approaches combine FI with machine learning methods [15], [16]. Approaches exist that focus on reducing the time of DNN reliability evaluation by reducing the number of FIs and combine the results with machine learning. For instance, a percentage of the parameters is selected for FI, and a set of features (absolute value, gradient, calculation time and layer location) is extracted and used with random forest regression for the vulnerability estimation [16]. However, such approaches do not leverage the obtained results in order to cost-effectively protect the DNNs. Other existing approaches focus on timing errors, e.g., first-order Taylor expansion approximates the weight sensitivity to design a DNN where sensitive weights are mapped

A 2-page abstract of this paper has been accepted at DATE2023 [1]

to robust multiple-and-accumulate operators [17]. Other studies proposed the use of dedicated low-overhead correction codes to protect DNN parameters [18]–[22]. However, they either apply the protection to all the NN parameters - regardless of their criticality - or rely on a modified training procedure. This can be problematic especially when dealing with DNNs with elevated training time and costs (weeks/months for industry-level production models, e.g., alphaGO [23], GPT-3 [24]) and with billions of parameters.

This work extends the state-of-the art by proposing for the first time a comprehensive framework to (i) evaluate the resilience of large DNNs to faults impacting their parameters, (ii) estimate the criticality of all the DNN parameters and their bits, and (iii) use this information for cost-effective DNN protection. More precisely, the proposed framework performs, as a first step, a datatype-and-layer-based FI, driven by the characteristics of the targeted DNN model. As a second step, it uses the results of FI to train ML classification models. This allows predicting the bit-accurate criticality of all network parameters. Last, ECC codes are selectively inserted to protect the critical bits and critical parameters, achieving low-overhead fault tolerance. Thanks to the proposed framework, we explored eight Convolutional Neural Networks and show memory savings up to 83% w.r.t. protecting all the network parameters with common ECC. Our framework is applied at post-training phase, and thus, does not require any special fault-aware training procedure.

The rest of the paper is organized as follows. Section II describes the proposed resilience evaluation and protection framework. Section III presents the results for eight case studies and Section IV draws conclusions.

II. Proposed Framework

Figure 1 depicts the overview of the proposed framework, which consists of three main steps, described below.

Step I: datatype-and-layer-based fault injection campaign

The first step of the proposed framework consists in performing a reduced number of targeted FIs, while keeping high the confidence of the FI outcome. This is achieved by considering a datatype-and-layer-based FI approach, to take into account the characteristics of the DNN model under study. DNNs consist of several layers, performing different operations, thus having different impact on the output. To capture the individual impact of the DNN layers, in this paper, we perform a layer-based simulation-based software FI, ensuring that a statistically significant number of random faults is injected in each DNN layer. We target the parameters of the DNN stored in the memory. Similar approaches were previously adopted to study the statistical resilience of DNNs (e.g. in [5], [13]).

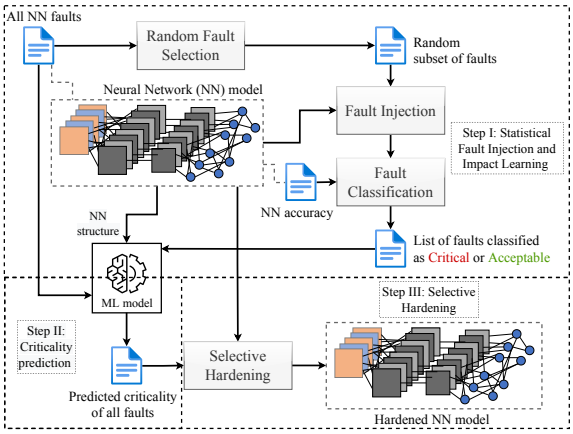

Fig. 1: Overview of the proposed framework.

Furthermore, faults striking different bits of a given DNN parameter impact the DNN output in different ways [13]. Their criticality depends on the datatype and the position of the impacted bit in the DNN under study. Usually, the DNN training process generate parameters having values in a relatively small range [13], [21]. Therefore, faults impacting data encoding approaches allowing large value ranges are more likely to generate out-of-scale values. For example, using floating-point datatype has a particularly negative effect for fault tolerance, since the possible data range is very extended. On the other hand, using fixed-point datatype allows the range of values to be reduced, mitigating the fault effect in DNNs [5], [13], [21].

In this paper, as fault model we consider the Single Event Upset (SEU), which can be caused by a single energetic particle (e.g. cosmic rays and high energy protons) and is a soft (non-destructive) error [25]. To mimic this condition, we inject random faults in DNN parameters, according to the *statistical fault injection* formula proposed in [26]: $fault_injections = \frac{N}{1+e^2 \times \frac{N-1}{t^2 \times 0.25}}$, where N is the number of NN parameters, e is the desired error margin (1% in our case), and t depends on the desired confidence level (we used t=2.5758 to achieve 99% confidence level). The FI procedure is done once and results can be used as many times as necessary in the next steps. The use of the approach in [26] applied to DNNs has also been validated recently in [27], where the authors showed that it is necessary to apply the aforementioned formula to each layer of the DNN to obtain meaningful results. Therefore, as already mentioned, we assess the fault tolerance of every DNN layer through statistical FI. However, performing exhaustive injection may be feasible for layers having few parameters. Figure 2 shows the necessary injections in both statistical and exhaustive approaches as a function of layer parameters, along with the relative percentage of injection savings obtained with the statistical approach. For example, when a layer has 4164 parameters, the statistical FI allows performing 3331 injections, thus avoiding 20% of the injections. However, the user may be willing to pay the extra 20% in order to have an exhaustive characterization and not a ML-based prediction. Conversely, for big convolutional layers with a lot more parameters, it is simply not feasible to inject exhaustively. For example, the fourth layer of ResNet-18 has convolutional layers with 2,359,296 parameters; the statistical injection approach allows reducing the necessary injections to 16,525 faults (>99% savings) to characterize the layer criticality with 1% error and 99% confidence. The proposed framework lets the user choose the

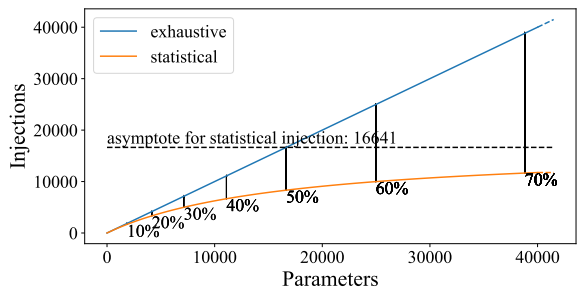

Fig. 2: Necessary injections for statistical VS exhaustive approaches (according to [26]), with percentage reduction.

percentage of injection savings under which a statistical FI is not deemed necessary because an exhaustive one is possible according to the available resources.

Moreover, we inject on different bits of the selected parameters, depending on their datatype and width. For Floating-Point (FP) data, we inject the sign bit, linearly spaced exponent bits and linearly spaced mantissa bits. For Fixed-Point (FxP), we inject linearly spaced integer bits and linearly spaced fractional bits. To assess the impact of an injected fault, we execute the whole NN test set (i.e., we perform all the inferences) and compare the obtained results with the golden reference (i.e., the fault-free NN). If the fault has a critical impact w.r.t. a user-defined metric (e.g. top-1 accuracy reduced more than a given value), we classify as critical both the injected bit and the parameter it belongs to. The framework user can choose how many and which bits to inject. In this paper, we kept the number of injected bits per parameter lower than 10. Then, as sketched in Figure 3, we consider the non-injected bits in a given parameter as critical if they have a left-hand-side (i.e., more significant) bit classified as critical. If a non-injected bit has a left-hand-side bit classified as *acceptable* (i.e., the impact of the fault was not critical), it is considered as acceptable. For FP, we apply this approach to both exponent and mantissa. For NNs having parameters encoded with low bit width data (e.g., 8 bits), we inject in every bit.

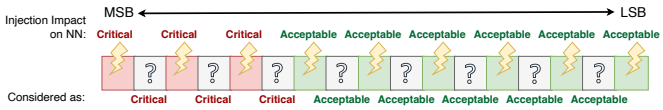

Fig. 3: Classification injected and non-injected parameter bits.

Step II: ML-based parameter and bit criticality prediction

In the second step, the framework predicts the criticality of all NN parameters and bits where FI was **not performed**. To do so, we train an ML-classification model using as input training data the FI results obtained from step I. Within the proposed framework, the user can choose the ML model. For our experiments, in this paper we used the Random Forest (RF) model, which exhibits high degree of robustness to over-fitting and has efficient execution time even for large datasets. A similar choice was made in previous work, where an RF-based regression model was used [16], instead of classification. Furthermore, we will show that the RF model is sensible to imbalanced datasets obtained for some NN configurations; thus, we used also a balanced version of RF, i.e., Balanced Random Forest (BRF).

Step III: Selective hardening

After steps I and II, the proposed framework is able to estimate the criticality of each parameter and bit. Step III uses this information to apply efficient selective protection mechanisms and explore their effects. In this paper, we propose and evaluate a set of mechanisms, based on selective Hamming ECC (single bit correction), to cost-effectively protect the DNN:

1) ECC protecting all the bits of the Critical Parameters (CP).
2) ECC only protecting the Critical Bits of the Critical Parameters (CBCP). This category is further divided into two:
   a) each critical parameter has a dedicated ECC (CBCP-Single, or CBCP-S).
   b) an ECC code is produced for all critical bits of a block of parameters (CBCP-Block or CBCP-B).

For the last two categories, the framework also proposes the option to store the ECCs in the LSBs of the critical parameters themselves - to further reduce the area overhead - and explores whether this option has a negative impact on the NN result quality.

For all the above mechanisms, additional memory is required to store the information about the criticality of NN parameters. The framework uses a *classification signature*: an array of bits - each one corresponding to an NN parameter - indicating whether the parameter has been classified as critical or not. Such classification signature could be further compressed with existing compression algorithms, to reduce memory. Finally, the classification signature needs to be protected with ECC to prevent faults from impacting the classification. For a coarse-grain ECC the overhead is negligible for big signatures. Indeed, using an m-bit Hamming ECC guarantees protection to $2^m - m - 1$ bits. For example, a signature vector of 1Mbits can be protected with only 20 bits of ECC code. Alternatively, the Hamming ECC for the classification signature can also be applied with high granularity, e.g. using 7 bits per 64-bit signature chunk would result in a final ECC with an overhead of 0.34% for a 32-bit encoded NN. It is up to the user to choose the most suitable configuration, according to the requirements.

For the CBCP mechanism, different parameters may have different number of critical bits. The user can choose whether to store this information for each critical parameter, along with the classification signature, or to protect the same bits for all parameters within a layer and store only this information (i.e., which bits to protect for a given layer). In the second case, the framework protects a given bit in a layer if it is critical for at least one parameter in that layer. In this paper, we adopt the latter approach. As each layer is dedicated to a specific functionality, the parameters within a layer are expected to have similar characteristics, and thus, not large variations in the number of critical bits.

III. Use cases for the proposed framework

We used the proposed framework to assess the fault tolerance and then protect two Convolutional Neural Networks, LeNet-5 and ResNet-18, each in four versions: (i) 32-bit floating-point (FP32), (ii) 32-bit fixed-point with 31 bits for fractional part (FxP32), (iii) 16-bit fixed-point with 15 bits for fractional part (FxP16), and (iv) 8-bit fixed-point with 7 bits for fractional part (FxP8), for a total of eight use cases. All versions of LeNet-5 were trained with the MNIST dataset and all versions of ResNet-18 were trained with the CIFAR10 dataset with an NVIDIA Quadro RTX 5000 GPU. The top-1 accuracy values of the NNs are reported in Table I, along with the training time. The FxP versions were trained using a quantization-aware training process [28].

In step I, we performed a statistical FI of random bit flips. We injected, on average, 292,000 faults for LeNet-5 NNs and 2,100,000 faults for ResNet-18 NNs. The statistical FI is done only once per NN; thanks to a highly parallel computing grid, we kept the FI time reasonably low: it took us on average one hour to realize the statistical FI on LeNet-5 NNs and two days on ResNet-18 NNs.

TABLE I: Top-1 accuracy and training time of the NNs under study

	FP32	**FxP32**	**FxP16**	**FxP8**
LeNet-5 top-1 accuracy	99.02%	98.7%	98.95%	98.81%
ResNet-18 top-1 accuracy	92.59%	93.8%	93.59%	93.18%
LeNet-5 training time(s)	1220	984	870	889
ResNet-18 training time(s)	pre-trained [29]	23989	23824	23929

In step II, two ML classification models are used (i) Random Forest (RF) and (ii) Balanced Random Forest (BRF). We used as ML input features only structural properties of the NNs, in order to be able to apply the proposed approaches in a post-training phase, unlike previous studies [21], [22]. In particular, the used features are the injected parameter's sign, the position of the injected bit in the parameter, the injected NN layer, and the output feature map and filter channel affected by the fault, depicted in Figure 4. The ML models are evaluated with a k-fold cross

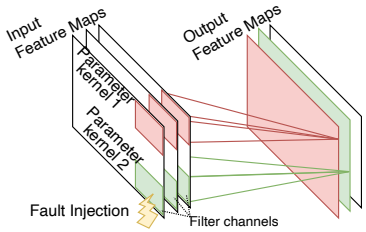

Fig. 4: Affected output feature map and filter channel.

validation process with k=10. The dataset ratio used to train/test the ML models is 70%/30%. As stated in Section II, the classification into *critical* and *acceptable* faults depends on a user-defined metric which provides the tolerated output quality degradation. Thus, we trained the ML models with 20 different tolerated top-1 accuracy degradation values, from 0.5 to 10 with step 0.5. The average training time for a ML model was lower than one hour, for all experiments.

In step III, we set at 120 the dimension of the block of parameters used in the CBCP-B approach. This allows a block of 120 parameters to be protected with only 7 ECC bits when having one critical bit, with 8 ECC bits when having 2 critical bits, and so on; indeed, as already mentioned, a m-bit Hamming ECC guarantees protection to $2^m - m - 1$ critical bits. In the interest of conciseness, in the next subsections we present a subset of the results per step.

Step I: datatype-and-layer-based fault injection campaign

This subsections reports the results of the FI campaigns. Table II shows the percentage of bits that have been classified as acceptable and as critical for all the explored networks as a result of the datatype-and-layer-based FI campaigns. As the trend is very clear and for the sake of conciseness, we report half of the results, omitting one every two. Overall, we observe that, when the network parameters are encoded using the FxP datatype, the impact of fault is less critical than in the FP case. As mentioned also in Section II, this happens because faults impacting data encoding approaches allowing narrow value ranges are less likely to generate out-of-scale values.

TABLE II: Fault Injection results.

Tolerated Top1 Acc. Reduction	FP32		FxP32		Fx16		FxP8	
	Accep.	Critic.	Accep.	Critic.	Accep.	Critic.	Accep.	Critic.
LeNet-5								
0.50%	98.23%	1.77%	99.21%	0.79%	99.71%	0.29%	99.70%	0.30%
1.50%	98.36%	1.64%	99.80%	0.20%	99.94%	0.06%	99.95%	0.05%
2.50%	98.38%	1.62%	99.91%	0.09%	99.98%	0.02%	99.98%	0.02%
3.50%	98.39%	1.61%	99.95%	0.05%	99.99%	0.01%	99.99%	0.01%
4.50%	98.40%	1.60%	99.98%	0.02%	100%	0.00%	99.99%	0.01%
5.50%	98.41%	1.59%	99.99%	0.01%	100%	0.00%	99.99%	0.01%
6.50%	98.42%	1.58%	99.99%	0.01%	100%	0.00%	100%	0.00%
7.50%	98.43%	1.57%	100%	0.00%	100%	0.00%	100%	0.00%
8.50%	98.44%	1.56%	100%	0.00%	100%	0.00%	100%	0.00%
9.50%	98.44%	1.56%	100%	0.00%	100%	0.00%	100%	0.00%
ResNet-18								
0.50%	91.56%	8.44%	95.38%	4.62%	95.56%	4.44%	95.78%	4.22%
1.50%	91.58%	8.42%	96.91%	3.09%	97.02%	2.98%	97.98%	2.02%
2.50%	91.64%	8.36%	97.61%	2.39%	97.66%	2.34%	98.62%	1.38%
3.50%	91.70%	8.30%	98.01%	1.99%	98.05%	1.95%	98.94%	1.06%
4.50%	91.74%	8.26%	98.28%	1.72%	98.31%	1.69%	99.14%	0.86%
5.50%	91.76%	8.24%	98.50%	1.50%	98.52%	1.48%	99.28%	0.72%
6.50%	91.78%	8.22%	98.68%	1.32%	98.69%	1.31%	99.39%	0.61%
7.50%	91.79%	8.21%	98.83%	1.17%	98.84%	1.16%	99.47%	0.53%
8.50%	91.80%	8.20%	98.96%	1.04%	98.96%	1.04%	99.54%	0.46%
9.50%	91.80%	8.20%	99.07%	0.93%	99.07%	0.93%	99.60%	0.40%

The reported numbers are referred to bits classified as Critical or Acceptable

Step II: Machine-learning parameter & bit criticality prediction

Firstly, we deem necessary to validate the effectiveness of the prediction w.r.t. real criticality, since the approach is build on top of statistical fault injection. To do so, we performed an exhaustive fault injection of all parameters of the FP32 version of LeNet and compared the results with the prediction of RF and BRF models, trained on statistical fault injection results. The results of such experiment showed that the difference between the number of real-critical parameters and the number of parameters predicted as critical is on average 0.09% (min 0.03%, max 0.17%) when RF is used, and 0.29% (min 0.12%, max 0.49%) when BRF is used - well below the 1% error mentioned in Section II. Moreover, this difference is translated in the models always predicting more critical parameters than the exhaustive approach. This means that the predictions are conservative, i.e., we do not leave critical parameters unprotected but may protect some non-critical ones, especially with BRF. For example, with tolerate top-1 accuracy reduction of 0.5%, 52.86% of the parameters are critical (exhaustive), the RF prediction is 52.94% and the BRF is 53.12%. Thus, we can conclude that the prediction model trained on statistical fault injections delivers reliable criticality predictions. Therefore, from this point forward, we use conventional ML metrics to validate the models. Also, performing exhaustive FI on bigger NN, such as ResNet-18, is not feasible - which stresses the need of the proposed approach.

In Table III, we report prediction results regarding the criticality of the parameters' bits, for the FP32 and FxP8 datatypes. As done in Table II, we reduce the number of results, omitting one every two from 3.0% forward. For the other NN versions, the general trend is similar, thus we do not report all the results, for the sake of conciseness. We report FP32 to show that the RF model is sufficient to perform a correct prediction. For FxP data, a BRF model is sometimes necessary. We report FxP8 data to show the merit of the approach also when dealing with a low-precision data encoding. In the table, we report the *Area under the ROC Curve (ROC AUC)* metric and the ML prediction results (percentages of correctly/incorrectly predicted classes) ROC AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0. An overall observation is that our approach is able to provide accurate results for the classification of the parameters and their bits, given the general high values of ROC AUC. Note that, when n/a is reported, it means that in the experiment the critical faults were insufficient to (i) perform the model cross-validation (Mean ROC AUC columns) and (ii) perform the ML training (other columns).

When the network parameters are encoded with FxP, we observe a high percentage of mispredicted critical bits (compare columns 11 and 12). This is also highlighted by the *Recall* metric for Critical faults reported in the table. Recall measures between 0 and 1 the correct coverage of the class. High values mean that the critical faults are rarely predicted as Acceptable. This issue is likely due to the imbalanced number of samples in the two classes (i.e. few critical and lots of acceptable). To deal with this issue, we resorted to the balanced version of RF, highly reducing the percentage of

TABLE III: ML-based fault prediction model. Results are referred to bits of NN parameters.

Tolerated Top1 Acc. Reduction	FP32 Mean ROC AUC	FP32 True Class:Acceptable Predicted Acceptable	FP32 True Class:Acceptable Predicted Critical	FP32 True Class:Critical Predicted Acceptable	FP32 True Class:Critical Predicted Critical	FP32 Critical Recall	FxP8 with RF Mean ROC AUC	FxP8 with RF True Class:Acceptable Predicted Acceptable	FxP8 with RF True Class:Acceptable Predicted Critical	FxP8 with RF True Class:Critical Predicted Acceptable	FxP8 with RF True Class:Critical Predicted Critical	FxP8 with RF Critical Recall	FxP8 with BRF Mean ROC AUC	FxP8 with BRF True Class:Acceptable Predicted Acceptable	FxP8 with BRF True Class:Acceptable Predicted Critical	FxP8 with BRF True Class:Critical Predicted Acceptable	FxP8 with BRF True Class:Critical Predicted Critical	FxP8 with BRF Critical Recall
LeNet-5																		
0.5%	0.9965	98.17%	0.07%	0.03%	1.74%	0.98	0.9742	99.63%	0.07%	0.20%	0.10%	0.33	0.9951	93.59%	6.11%	0.00%	0.30%	1.00
1.0%	0.9985	98.29%	0.04%	0.02%	1.66%	0.99	0.9762	99.84%	0.06%	0.07%	0.02%	0.25	0.9958	96.75%	3.15%	0.00%	0.10%	1.00
1.5%	0.9994	98.33%	0.03%	0.01%	1.63%	0.99	0.9583	99.95%	0.00%	0.04%	0.00%	0.05	0.9917	97.78%	2.17%	0.00%	0.05%	1.00
2.0%	0.9997	98.36%	0.02%	0.01%	1.61%	0.99	0.9639	99.97%	0.00%	0.03%	0.00%	0	0.9942	98.72%	1.25%	0.00%	0.03%	1.00
2.5%	0.9997	98.36%	0.02%	0.01%	1.60%	0.99	0.9535	99.98%	0.00%	0.02%	0.00%	0	0.9987	97.81%	2.17%	0.00%	0.02%	1.00
3.0%	0.9997	98.36%	0.02%	0.01%	1.60%	0.99	0.9493	99.99%	0.00%	0.01%	0.00%	0	0.9984	96.44%	3.54%	0.00%	0.01%	1.00
4.0%	0.9998	98.38%	0.02%	0.02%	1.58%	0.99	0.9747	99.99%	0.00%	0.01%	0.00%	0	0.9983	95.91%	4.08%	0.00%	0.01%	1.00
5.0%	0.9999	98.38%	0.02%	0.02%	1.57%	0.99	0.9998	99.99%	0.00%	0.01%	0.00%	0	0.9983	95.52%	4.47%	0.00%	0.01%	1.00
6.0%	0.9999	98.40%	0.01%	0.02%	1.57%	0.99	n/a	99.99%	0.00%	0.01%	0.00%	0	nan	94.65%	5.35%	0.00%	0.01%	1.00
7.0%	0.9999	98.40%	0.02%	0.02%	1.56%	0.99	n/a	100.00%	0.00%	0.00%	0.00%	0	nan	84.39%	15.61%	0.00%	0.00%	1.00
8.0%	0.9999	98.42%	0.01%	0.01%	1.56%	0.99	n/a	100.00%	0.00%	0.00%	0.00%	0	nan	84.39%	15.61%	0.00%	0.00%	1.00
9.0%	0.9999	98.42%	0.01%	0.01%	1.55%	0.99	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
10.0%	0.9998	98.44%	0.01%	0.00%	1.55%	1.00	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
ResNet-18																		
0.5%	0.9999	91.53%	0.02%	0.01%	8.43%	1.00	0.9965	95.41%	0.37%	0.48%	3.74%	0.89	0.9978	92.87%	2.91%	0.05%	4.17%	0.99
1.0%	0.9999	91.54%	0.03%	0.01%	8.42%	1.00	0.9968	96.98%	0.30%	0.47%	2.26%	0.83	0.9979	94.21%	3.06%	0.02%	2.71%	0.99
1.5%	0.9999	91.55%	0.03%	0.01%	8.41%	1.00	0.9967	97.80%	0.18%	0.41%	1.61%	0.80	0.9980	94.85%	3.12%	0.01%	2.01%	0.99
2.0%	0.9999	91.56%	0.05%	0.02%	8.37%	1.00	0.9965	98.24%	0.13%	0.32%	1.31%	0.80	0.9982	95.31%	3.06%	0.01%	1.62%	0.99
2.5%	0.9999	91.56%	0.07%	0.03%	8.34%	1.00	0.9963	98.50%	0.12%	0.25%	1.13%	0.82	0.9983	95.68%	2.95%	0.01%	1.37%	1.00
3.0%	0.9999	91.60%	0.08%	0.04%	8.29%	0.99	0.9957	98.66%	0.14%	0.22%	0.98%	0.81	0.9985	96.02%	2.77%	0.00%	1.20%	1.00
4.0%	0.9999	91.65%	0.07%	0.05%	8.23%	0.99	0.9955	98.92%	0.13%	0.17%	0.78%	0.82	0.9986	96.80%	2.25%	0.01%	0.95%	0.99
5.0%	0.9999	91.68%	0.07%	0.03%	8.22%	1.00	0.9950	99.10%	0.11%	0.16%	0.62%	0.80	0.9986	97.27%	1.94%	0.01%	0.78%	0.99
6.0%	0.9999	91.70%	0.08%	0.02%	8.21%	1.00	0.9958	99.26%	0.08%	0.16%	0.50%	0.76	0.9986	97.57%	1.77%	0.00%	0.65%	0.99
7.0%	0.9999	91.71%	0.08%	0.01%	8.20%	1.00	0.9959	99.35%	0.08%	0.15%	0.41%	0.73	0.9985	97.78%	1.65%	0.00%	0.57%	1.00
8.0%	0.9999	91.71%	0.08%	0.01%	8.19%	1.00	0.9963	99.44%	0.07%	0.14%	0.35%	0.71	0.9985	97.94%	1.57%	0.00%	0.49%	0.99
9.0%	0.9999	91.72%	0.08%	0.01%	8.19%	1.00	0.9961	99.52%	0.05%	0.13%	0.30%	0.69	0.9985	98.05%	1.52%	0.00%	0.43%	0.99
10.0%	0.9999	91.71%	0.10%	0.01%	8.18%	1.00	0.9952	99.58%	0.04%	0.12%	0.26%	0.67	0.9986	98.11%	1.51%	0.00%	0.38%	0.99

when n/a, the critical faults are insufficient to (i) perform the model cross-validation (Mean ROC AUC columns) and (ii) perform the ML training (other columns). See Table II.

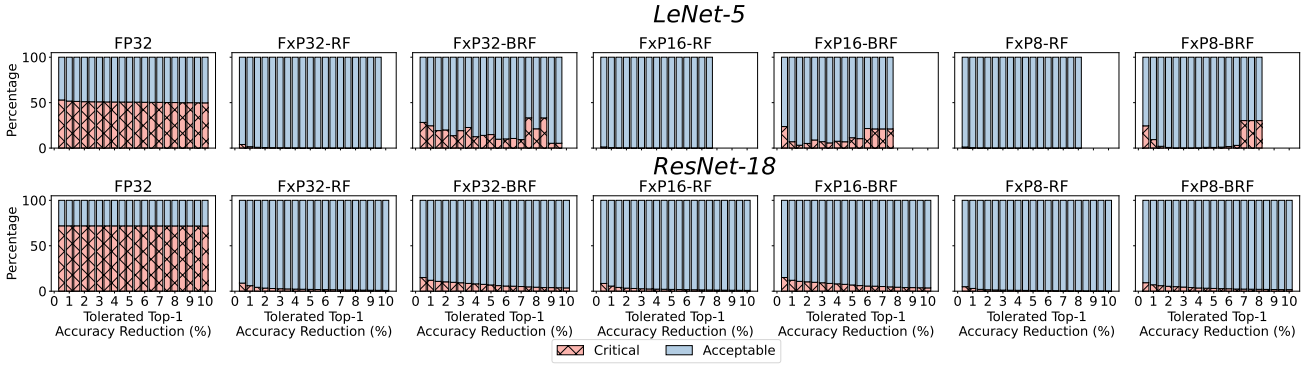

Fig. 5: ML prediction of Critical and Acceptable parameters for both LeNet-5 and ResNet-18

critical bits predicted as acceptable, as shown in column 17. For instance, for 0.5% top-1 accuracy reduction, LeNet-5 with FxP8 has 0.2009% critical bits predicted as acceptable (recall 0.33), when RF is used, and 0% when BRF is used (recall 1.00). ResNet-18 with FxP8 has 0.4814% critical bits predicted as acceptable (recall 0.89), when RF is used, and 0.0451% (recall 0.99) when BRF is used. As a drawback of using BRF, we highlight that, while the number of mispredicted critical bits is reduced, the percentage of mispredicted acceptable bits increases (column 16). Note that, this only entails a protection overhead and not a hazard in terms of reliability. Nonetheless, the framework user can choose which RF version to use according to final requirements. So far, we reported results concerning bit criticality. As already mentioned, we consider a parameter as critical when it contains at least one critical bit. Hence, in Figure 5, we report the ML prediction results regarding the total number of critical and acceptable parameters for all versions of the NNs and for all tolerated top-1 accuracy reductions that we explored. With FP representation, we obtain significantly higher number of critical parameters compared to the FxP versions of the networks, in line with results observed in previous works investigating the fault tolerance of DNNs [5], [13]. Furthermore, as BRF is more conservative, the number of parameters considered as critical is increased compared to the RF. For instance, for 0.5% top-1 accuracy reduction, the RF model predicted LeNet-5 with FxP32 having 1,745 critical parameters, while BRF predicted 12,477 (total 44,190); for ResNet with FxP32, RF predicted 979,577 critical parameters, while BRF predicted 1,654,378 (total 10,992,320).

Step III: Selective hardening

Table IV reports the results in terms of memory overhead for the proposed approaches (i.e., CP, CBCP-S, and CBCP-B) and compares them to the conventional ECC method, where ECC protects all parameters and all bits (AP). Results are reported for the case where an accuracy reduction of 0.5% was tolerated, thus the number of critical bits is high. Compared to AP, on average, CP achieves 61.22% reduction, CBCP-S 71.95% and CBCP-B 80.20%. As also reported in Section II, the classification signature can be protected with few bits of coarse-grain ECC - 16 bits for

TABLE IV: Memory overhead comparison

	Overhead					Overhead reduction w.r.t. AP		
	ECC				Class/tion Signature	Class/tion Signature and		
	AP	CP	CBCP-S	CBCP-B		CP	CBCP-S	CBCP-B
LeNet-5								
FP32	18.75%	9.926%	5.017%	0.126%	3.125%	30.396%	56.575%	82.662%
FxP32	18.75%	0.740%	0.363%	0.009%	3.125%	79.384%	81.399%	83.286%
FxP16	31.25%	0.383%	0.215%	0.009%	6.250%	78.773%	79.312%	79.972%
FxP8	50.00%	0.583%	0.351%	0.012%	12.500%	73.835%	74.298%	74.975%
ResNet-18								
FP32	18.75%	13.488%	4.572%	0.132%	3.125%	11.398%	58.947%	82.630%
FxP32	18.75%	1.671%	0.835%	0.019%	3.125%	74.422%	78.878%	83.234%
FxP16	31.25%	2.668%	1.601%	0.036%	6.250%	71.462%	74.877%	79.885%
FxP8	50.00%	2.476%	1.857%	0.046%	12.500%	70.049%	71.287%	74.907%

Coarse-grain ECC size to protect the Classification Signature for LeNet-5 is 16 bits and for ResNet-18 is 24 bits, negligible compared to the dimension of the NNs.

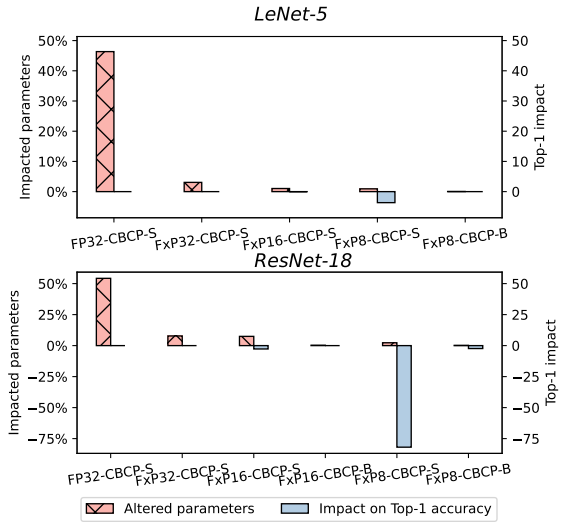

Fig. 6: Altered parameters and impact on Top1 accuracy.

LeNet-5 and 24 for ResNet-18 - or with very low overhead, e.g. 0.34% of fine-grain ECC overhead for a 32-bit encoded NN.

To further reduce the memory overhead, we explore the integration of the ECC obtained with the proposed approaches in the LSB of the critical parameters. In this case, the memory overhead comes only from the Classification Signature (i.e., the overhead for CP, CBCP-S/B in Table IV would be 0). However, this choice is likely to alter the value of the critical parameters, thus may affect the NN accuracy. Figure 6 shows the results of this approach in terms of percentage of the altered parameters and of how much this modification reduced the NN top-1 accuracy. While for 32-bit parameters the accuracy is not impacted (*F(x)P32-CBCP-S* in the figures), when reduced datatypes (especially 8 bits) are used, altering the LSBs of the parameters significantly impacts the accuracy, e.g., with a reduction of 3.65% for the FxP8 LeNet and 81.91% for FxP8 ResNet considering RF (*FxP8-CBCP-S* in the figures). Nonetheless, the results show that using the proposed CBCP-Block mechanism drastically reduces this limitation by altering a significantly lower amount of parameters and bits (*FxP16-CBCP-B* and *FxP8-CBCP-B* in the figures).

IV. Conclusions

We proposed a framework to enable the post-training fault tolerance characterization of all parameters of big DNNs, with the goal applying selective protection. This process is infeasible with conventional FI approaches, due to the dimensions of DNNs. The proposed framework is composed of three steps. The first one obtains data regarding the criticality of DNN parameters and bits through targeted FI campaigns. Then, in the second step, classification-based ML methods are trained with FI data and used to predict the criticality of all non-injected DNN parameters and bits. Using this information, the last step applies selective ECC to the DNN to cost-effectively protect it. Thanks to the proposed framework, the exploration of eight DNN models was possible. Results show memory savings up to 83% w.r.t. conventional ECC on LeNet-5 and ResNet-18 DNNs with different datatypes and widths.

References

[1] M. Traiola *et al.*, "A machine-learning-guided framework for fault-tolerant DNNs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[2] Y. LeCun *et al.*, "Deep learning," *Nature*, vol. 521, 2015.

[3] F. F. d. Santos *et al.*, "Analyzing and increasing the reliability of convolutional neural networks on gpus," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2019.

[4] C. Torres-Huitzil *et al.*, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17 322–17 341, 2017.

[5] A. Ruospo *et al.*, "Investigating data representation for efficient and reliable convolutional neural networks," *Microprocessors and Microsystems*, vol. 86, p. 104318, 2021.

[6] A. Lotfi *et al.*, "Resiliency of automotive object detection networks on gpu architectures," in *IEEE Int. Test Conference (ITC)*, 2019, pp. 1–9.

[7] A. Ruospo *et al.*, "Pros and cons of fault injection approaches for the reliability assessment of deep neural networks," in *2021 IEEE 22nd Latin American Test Symposium (LATS)*, 2021, pp. 1–5.

[8] A. Ruospo *et al.*, "A survey on deep learning resilience assessment methodologies," *Computer*, vol. 56, no. 2, pp. 57–66, 2023.

[9] A. Mahmoud *et al.*, "Pytorchfi: A runtime perturbation tool for DNNs," in *IEEE/IFIP Int. Conf. Dependable Systems and Networks Workshops (DSN-W)*, 2020, pp. 25–31.

[10] Z. Chen *et al.*, "Tensorfi: A flexible fault injection framework for tensorflow applications," *CoRR*, vol. abs/2004.01743, 2020.

[11] B. Salami *et al.*, "On the resilience of rtl nn accelerators: Fault characterization and mitigation," in *Int. Symp. Computer Architecture and High Performance Computing (SBAC-PAD)*, sep 2018, pp. 322–329.

[12] Y. He *et al.*, "Fidelity: Efficient resilience analysis framework for deep learning accelerators," in *IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2020, pp. 270–281.

[13] G. Li *et al.*, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Int. Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, 2017.

[14] C. Bolchini *et al.*, "Fast and accurate error simulation for cnns against soft errors," *IEEE Transactions on Computers*, pp. 1–14, 2022.

[15] A. Chaudhuri *et al.*, "Efficient fault-criticality analysis for ai accelerators using a neural twin," in *2021 IEEE International Test Conference (ITC)*, 2021, pp. 73–82.

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

[17] W. Choi *et al.*, "Sensitivity based error resilient techniques for energy efficient deep neural network accelerators," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[18] K. Huang *et al.*, "Functional Error Correction for Reliable Neural Networks," in *2020 IEEE International Symposium on Information Theory (ISIT)*, Jun. 2020, pp. 2694–2699.

[19] S. Burel *et al.*, "Zero-overhead protection for cnn weights," in *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2021, pp. 1–6.

[20] U. Zahid *et al.*, "Fat: Training neural networks for reliable inference under hardware faults," in *2020 IEEE International Test Conference (ITC)*, 2020, pp. 1–10.

[21] S.-S. Lee *et al.*, "Value-aware Parity Insertion ECC for Fault-tolerant Deep Neural Network," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Mar. 2022, pp. 724–729.

[22] N. Cavagnero *et al.*, "Fault-Aware Design and Training to Enhance DNNs Reliability with Zero-Overhead," 2022, arXiv:2205.14420.

[23] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.

[24] T. B. Brown *et al.*, "Language Models are Few-Shot Learners," Jul. 2020, arXiv:2005.14165 [cs].

[25] J. F. Ziegler *et al.*, "Effect of cosmic rays on computer memories," *Science*, vol. 206, no. 4420, pp. 776–788, 1979.

[26] R. Leveugle *et al.*, "Statistical fault injection: Quantified error and confidence," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2009, pp. 502–506.

[27] A. Ruospo *et al.*, "Assessing convolutional neural networks reliability through statistical fault injections," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[28] B. Jacob *et al.*, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," Dec. 2017, arXiv:1712.05877.

[29] "huyvnphan/PyTorch_cifar10," Jan. 2021. [Online]. Available: https://zenodo.org/record/4431043