# A machine-learning-guided framework for fault-tolerant 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. Nevertheless, DNN results may be incorrect, not only because of the network intrinsic inaccuracy, but also due to faults affecting the hardware. Ensuring the fault tolerance of DNN is crucial, but common fault tolerance approaches are not cost-effective, due to the prohibitive overheads for large DNNs. This work proposes a comprehensive framework to assess the fault tolerance of DNN parameters and cost-effectively protect them. As a first step, the proposed framework performs a statistical fault injection. The results are used in the second step with classification-based machine learning methods to obtain a bit-accurate prediction of the criticality of all network parameters. Last, Error Correction Codes (ECCs) are selectively inserted to protect only the critical parameters, hence entailing 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 79% memory w.r.t. conventional ECC approaches.**

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

## I. Introduction

Deep Neural Networks (DNNs) are currently one of the most intensively and widely used predictive models in the field of Machine Learning (ML). Although not always 100% accurate, recent models reach incredible accuracy levels. However, their results can be affected by hardware faults. The sources of faults are various, e.g., escaped post-fabrication faults, silicon ageing, and environmental stress, such as heat, humidity, vibration, and radiation [1]. Although DNN models are inherently resilient, faults can still occur after training and have a detrimental effect on the application [2], [3]. Common fault tolerance approaches, e.g. Triple Modular Redundancy (TMR) and Error Correction Codes (ECC), are not cost effective, if applied naively. Therefore, DNN most critical parts should be identified and protected. The evaluation of the DNN resilience is typically based on Fault Injection (FI). FI approaches are usually classified as simulation-based, hardware-based or radiation-based [4]. In general, exhaustive FI is not computationally feasible, given the high amount of possible injection points of complex systems, such as DNNs. Therefore, the majority of DNN resilience assessments are based on statistical FI. Such an approach allows assessing only the criticality of the injected parameters. However, to cost-effectively protect a DNN, the criticality of all parameters should be assessed. To address this limitation, recent approaches combine FI with machine learning methods [5]. However, the results are not used to cost-effectively protect the DNNs. Studies proposing the use of dedicated low-overhead correction codes to protect DNN parameters exist [6]–[8]. 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 with DNNs with elevated training time and costs (weeks/months for industry-level production models) 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 to faults of a set of parameters of large DNNs, (ii) build a prediction model to estimate the criticality of all the DNN parameters and their bits, and (iii) use this information for cost-effective DNN protection. Our framework is applied at post-training phase without requiring any special fault-aware training procedure.

## II. Proposed framework

The proposed framework consists of three main steps:

*Step I, statistical fault injection campaign:* performing a reduced number of FIs, while keeping high the confidence of the FI outcome. Hence, we apply a simulation-based software statistical FI approach [9] for each layer of the DNN model under study. We target the parameters of the DNN stored in the memory. The FI procedure is done once and results can be used as many times as necessary in the next steps.

*Step II, ML-based parameter & bit criticality prediction:* An ML-classification model is trained with the FI results of step I, as input training data. The framework predicts the criticality of all NN parameters and bits where FI was **not performed**. The proposed framework allows the user to choose the ML model.

*Step III: Selective hardening:* After steps I and II, the proposed framework is able to estimate the criticality of each parameter. Step III uses this information to apply efficient selective protection mechanisms and explore their effects. In this paper, we propose a mechanism to cost-effectively protect the DNN, i.e., we apply Hamming ECC (for single bit correction) only to Critical Parameters (CP). We evaluate it w.r.t. the memory overhead reduction w.r.t. ECC applied naively to all parameters. It is important to highlight that it is important to store in memory also the information about the criticality of NN parameters, i.e. result of step II. To this end, we resort to 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. Finally, we assume that the classification signature is protected (e.g. with ECC), to prevent faults from impacting the classification.

## III. Case studies

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, for a total of eight use cases: (i) 32-bit floating-point (FP32), (ii) 32-bit fixed-point (FxP32), (iii) 16-bit fixed-point (FxP16), and (iv) 8-bit fixed-point (FxP8) –

978-3-9819263-7-8/DATE23/© 2023 EDAA

all FxP with one bit for the integer part. All versions of LeNet-5 and ResNet-18 were trained respectively with the MNIST and CIFAR10 datasets and reached >98.5% and >92.5% top-1 accuracy, respectively. All training were done with an NVIDIA Quadro RTX 5000 GPU. The FxP versions were trained using a quantization-aware training process [10]. In step I, as fault model we consider the Single Event Upset (SEU), which models the impact of a single energetic particle (e.g. cosmic rays and high energy protons) and is a soft (non-destructive) error [11]. To mimic this condition, we performed a statistical FI of random bit flips to obtain an assessment with 1% error margin and 99% confidence according to [9]. We injected, on average, 292,000 faults for LeNet-5 NNs and 2,100,000 faults for ResNet-18 NNs. In step II, we used two ML classification models (i) Random Forest (RF) and (ii) Balanced Random Forest (BRF). These models exhibit high degree of robustness to overfitting and have efficient execution time even for large datasets. In the experiments, we show that the RF model is sensible to imbalanced datasets, hance the use of BRF. We used as ML input features only structural properties of the NNs, to be able to apply the proposed approaches in a post-training phase. 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. The ML models are evaluated with a k-fold cross validation process with k=10. The dataset ratio used to train/test the ML models is 70%/30%. The classification into *critical* and *acceptable* faults depends on a user-defined metric which provides the tolerated output quality degradation. Thus, we trained the RF and BRF 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.

TABLE I: 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% |
| 10.00% | 98.45% | 1.55% | 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% |
| 10.00% | 91.81% | 8.19% | 99.12% | 0.88% | 99.12% | 0.88% | 99.62% | 0.38% |

*Step I:* Table I shows the percentage of bits that have been classified as acceptable and as critical as a result of the FI campaigns. For conciseness, we report the minimum and maximum values of tolerated top-1 accuracy reduction. 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. This happens because faults impacting data encoding approaches allowing narrow value ranges (as FxP) are less likely to generate out-of-scale values [3].

*Step II:* In Table II, we report prediction results regarding the criticality of the parameters' bits, for the FP32 and FxP8 datatypes. For the other NN versions, the general trend is similar. For conciseness, we report the minimum and maximum values of tolerated top-1 accuracy reduction for which critical faults were still present, hence the model training was possible. 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). Overall, our approach is generally able to provide accurate results for the classification of the parameters and their bits, as shown by the general high values of ROC AUC. However, when

TABLE II: ML-based fault prediction model.

| Datatype and ML model | Tolerated Top1 Acc. Reduction | Mean ROC AUC | True Class: Acceptable | | True Class: Critical | |
|---|---|---|---|---|---|---|
| | | | Predicted Acceptable | Predicted Critical | Predicted Acceptable | Predicted Critical |
| LeNet-5 | | | | | | |
| FP32 | 0.5% | 0.9965 | 98.16% | 0.06% | 0.03% | 1.73% |
| and RF | 5.5% | 0.9999 | 98.3950% | 0.0135% | 0.0166% | 1.5749% |
| FxP8 | 0.5% | 0.9742 | 99.62% | 0.07% | 0.20% | 0.09% |
| and RF | 5.5% | 0.9998 | 99.9938% | 0.0000% | 0.0062% | 0.0000% |
| FxP8 | 0.5% | 0.9951 | 93.59% | 6.10% | 0.00% | 0.30% |
| and BRF | 5.5% | 0.9999 | 98.3950% | 0.0135% | 0.0166% | 1.5749% |
| ResNet-18 | | | | | | |
| FP32 | 0.5% | 0.9999 | 91.53% | 0.02% | 0.01% | 8.43% |
| and RF | 10.0% | 0.9999 | 91.71% | 0.09% | 0.01% | 8.18% |
| FxP8 | 0.5% | 0.9965 | 95.40% | 0.37% | 0.48% | 3.73% |
| and RF | 10.0% | 0.9952 | 99.57% | 0.04% | 0.12% | 0.25% |
| FxP8 | 0.5% | 0.9978 | 92.87% | 2.91% | 0.04% | 4.17% |
| and BRF | 10.0% | 0.9986 | 98.11% | 1.50% | 0.00% | 0.37% |

ROC AUC = 0.0 $\implies$ 100% wrong predictions; ROC AUC = 1.0 $\implies$ 100% correct predictions

the network parameters are encoded with FxP, we observe a high percentage of critical bits mispredicted by RF model (compare columns 5 and 6). 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 BRF, highly reducing the percentage of critical bits predicted as acceptable. As a drawback of using BRF, we highlight that the percentage of mispredicted acceptable bits increases (column 4). Note that, this only entails a protection overhead and not a hazard in terms of reliability. The framework user can choose which RF version to use according to final requirements.

*Step III:* Table III reports the results in terms of memory overhead for the proposed protection approach (i.e., protect only Critical Parameters (CP) with Hamming ECC) and compares it to the conventional ECC method, where ECC protects all parameters (AP). Results are reported for the case where an accuracy reduction of 0.5% was tolerated, thus the number of critical parameters is high. Compared to AP, CP achieves up to 79.38% savings, and 61.22% on average.

TABLE III: Memory overhead comparison

| | ECC AP | ECC CP | Sign. | CP+Sign. gain vs AP | ECC AP | ECC CP | Sign. | CP+Sign. gain vs AP |
|---|---|---|---|---|---|---|---|---|
| | LeNet-5 | | | | ResNet-18 | | | |
| **FP32** | 18.75% | 9.926% | 3.125% | 30.396% | 18.75% | 13.488% | 3.125% | 11.398% |
| **FxP32** | 18.75% | 0.740% | 3.125% | 79.384% | 18.75% | 1.671% | 3.125% | 74.422% |
| **FxP16** | 31.25% | 0.383% | 6.250% | 78.773% | 31.25% | 2.668% | 6.250% | 71.462% |
| **FxP8** | 50.00% | 0.583% | 12.500% | 73.835% | 50.00% | 2.476% | 12.500% | 70.049% |

## REFERENCES


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

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

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

[4] ——, "Pros and cons of fault injection approaches for the reliability assessment of deep neural networks," in *IEEE LATS*, 2021, pp. 1–5.

[5] Y. Zhang *et al.*, "Estimating vulnerability of all model parameters in dnn with a small number of fault injections," in *IEEE ACM DATE*, 2022, pp. 60–63.

[6] K. Huang *et al.*, "Functional Error Correction for Reliable Neural Networks," in *IEEE ISIT*, Jun. 2020, pp. 2694–2699.

[7] S.-S. Lee *et al.*, "Value-aware Parity Insertion ECC for Fault-tolerant Deep Neural Network," in *IEEE ACM DATE*, Mar. 2022, pp. 724–729.

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

[9] R. Leveugle *et al.*, "Statistical fault injection: Quantified error and confidence," in *IEEE ACM DATE*, 2009, pp. 502–506.

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

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