A Benchmark Suite to Evaluate DNN's Resilience

Cristiana Bolchini[1], Alberto Bosio[2], Luca Cassano[1], Antonio Miele[1],
Salvatore Pappalardo[2], Dario Passariello[1], Annachiara Ruospo[3],
Ernesto Sanchez[3], Matteo Sonza Reorda[3], Vittorio Turco[3]

[1]*Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Milano, Italy*
[2]*Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, 69130 Ecully, France*
[3]*Politecnico di Torino, Dip. di Automatica e Informatica, Torino, Italy*

***Abstract*—Assessing AI systems reliability is essential before deploying them in safety-critical applications. While recent efforts have focused on improving model resilience to random hardware faults, meaningful comparison remains difficult due to the lack of standardized reference models. Different authors use different implementations, which makes comparisons unfair and biased: resilience is influenced by the training processes, the software framework, and data representations. To address these issues, this work introduces a benchmark suite of CNN models to test the resilience of DNNs. The benchmark is structured on different axes: software framework, hardware platform, data representation, task and dataset. It is aimed at providing a shared foundation for fair and reproducible resilience evaluation.**

***Index Terms*—Benchmark, Deep Learning, Fault Injection, Machine Learning, Reliability.**

I. Introduction

Deep Learning (DL) models, especially Convolutional Neural Network (CNNs), are increasingly integrated into safety-critical systems such as autonomous vehicles and medical diagnostics [1]. However, their reliability under hardware faults is a pressing concern [2]. Unlike traditional benchmarks focusing on accuracy or performance, resilience benchmarks assess how well models tolerate faults, which is critical for deployment in high-risk domains. Existing approaches lack standardized reference models, making fair comparisons difficult.

To bridge this gap, the authors propose a benchmark suite that provides standardized CNN implementations along with associated training/test datasets and fault injection results. The suite enables repeatable and comparable resilience assessments and supports hardening efforts.

II. Characterization Elements

When designing a CNN, various factors influence not only its functional performance but also its resilience to hardware faults. This section outlines those key elements, which serve as the basis for characterizing each benchmark. Please note that the current framework includes all relevant aspects, nevertheless it can be extended to accommodate additional considerations in the future. The benchmark suite characterizes each CNN model using the following axes:

- Software Framework: PyTorch [3] and TensorFlow [4] are supported. Other examples of software frameworks are Jax [5], TinyGrad [6] and litert [4]. Although functionally equivalent, framework-specific internal implementations may affect resilience. In section IV this aspect is shown.
- Hardware Platform: includes CPU, GPU, TPU, FPGA, Systolic arrays, and ASICs, each impacting fault behavior differently.
- Task: models are grouped by task—image classification, segmentation, object detection, and regression.
- Dataset: Complementing the task, this axis covers datasets dedicated to a specific task. In this poster there are CIFAR-10, CIFAR-100, GTSRB, and PASCAL VOC. The datasets influence training dynamics and fault impact.
- Training Process: a CNN needs a recipe to be trained. This axis covers precisely this aspect, since hyper-parameters like the optimizer, the learning rate, the number of epochs, and data augmentation do influence model resilience.
- Data Representation: finally, recent research proposed quantization methods by using simpler data types [7] and it has been shown that it plays a role in the resilience [8]. This axes covers data representations (like FP32, FP16, INT8, bfloat) to be studied in terms of resilience.

III. Proposed Benchmark Suite

The suite consists of 16 CNN models across 11 architectures (namely ResNet, VGG, DenseNet, GoogLeNet, MobileNetV2, DeepLabV3), trained on the four datasets (CIFAR-10, CIFAR-100, GTSRB and COCO). Both PyTorch and TensorFlow versions are included. Conversion from PyTorch to TensorFlow was done using Nobuco [9], ensuring consistent architecture and comparable accuracy.

In order to ensure reproducibility, the benchmark includes the architecture of the model, along with the training recipe; the fault injection list used in the experiments described

in the following; the results obtained in the aforementioned experiments.

IV. Reliability Assessment

To demonstrate the effectiveness of the benchmark, some experiments are reported in this section. The core methodology involves Statistical Fault Injection (SFI) targeting the trained weights using stuck-at faults [10]. Fault injection was conducted at the software level [11] across all layers (convolutional and linear). A 99% confidence level and a 1–3% error margin guided fault sampling. The same faults were injected into both frameworks to ensure fair comparison [10].

A. Image Classification

The experiments show that models trained using Adam generally show better fault masking than those using SGD. Notably, using the Adam optimizer results in over 12% more masked faults and improved mitigation of SDC-1 faults.

In addition, another obstacle is the framework used for training. Another critical factor is the choice of identical neural architectures trained on different platforms may yield divergent outcomes. Prior studies [43, 44] demonstrate that variations in algorithms and libraries can directly affect model performance. Even with identical hyper-parameters, different frameworks can produce networks with distinct weight distributions and accuracy levels. PyTorch and TensorFlow yield small but non-negligible differences in fault sensitivity due to backend numerical handling and kernel-level differences.

For instance:

- ResNet20 (CIFAR-10) showed 41.65% masked faults in PyTorch versus 46.65% in TensorFlow.
- Differences in SDC-1 were small (0.01%), but differences in masking were more noticeable.

Please note that a fault can be categorized when the faulty output (produced by that fault) and the fault-free output given the same input, are compared together as following:

(a) Masked if they do not affect the final vector score at all,
(b) Non-critical if they have an impact on the final vector score, but they keep the correct prediction, and
(c) Silent Data Corruption (SDC-1) if they produce a wrong prediction, i.e., an application's failure [7].

B. Image Segmentation

Similar results were observed in the context of a different task. For segmentation tasks (e.g., DeepLabV3 on PASCAL VOC), customized metrics like pixel-level accuracy and mIoU were used, and resilience was categorized into masked, tolerable, and critical SDCs.

Specifically,

(a) Masked if the predicted pixel labels in the faulty and golden outputs are identical;
(b) Tolerable if less than 1% of pixels are modified between the faulty and golden outputs, with no new classes appearing or existing classes disappearing;
(c) Critical SDC if at least 1% of pixels are modified between the faulty and golden outputs, or new classes appear/disappear.

In this regard, the results show that the experiments carried out with PyTorch resulted in 79.9% of masked faults and 1.35% of critical SDCs, while using TensorFlow shows 88.69% of masked faults and 2.31% critical SDCs.

V. Discussion and Implication

The benchmark suite highlights that framework and training choices affect fault tolerance. Thus, it is necessary to make comparisons through the employment of standardized models to avoid misleading conclusions. This applies to both (i) hardening strategies to produce resilient hardware and (ii) analysis framework to fairly compare the results. This benchmark suite addresses this need of standardization and supports reproducibility.

VI. Conclusions

This benchmark suite fills a critical gap in AI system design, enabling standardized evaluation of DL model resilience. It provides:

- a consistent set of neural architectures,
- standardized fault injection producers,
- future-proof design by allowing structural improvements.

The suite is aligned with the goals of AI safety regulations (e.g., EU AI Act) and is publicly available on github at the link https://github.com/ReADLBench/dnn-benchmark, to support community-driven expansion. Future directions include broader tasks, data formats, and hardware targets.

References

[1] Y. Wang and C. S. Ho, "Artificial intelligence in safety-critical systems: A systematic review," 2021. DOI: 10.1108/IMDS-07-2021-0419. [Online]. Available: https://doi.org/10.1108/IMDS-07-2021-0419.

[2] *Regulation of the European parliament and of the council laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts*, https://eur-lex.europa.eu/resource.html?uri=cellar:e0649735-a372-11eb-9585-01aa75ed71a1.0001.02/DOC_1&format=PDF, Accessed: 2025-04-16.

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[4] Martín Abadi *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[5] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, *JAX: Composable transformations of Python+NumPy programs*, version 0.3.13, 2018. [Online]. Available: http://github.com/jax-ml/jax.

[6] tiny corp., *Https://github.com/tinygrad/tinygrad*.

[7] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. Conf. Computer Vision and Pattern Recognition*, 2018.

[8] A. Ruospo, E. Sanchez, M. Traiola, I. O'connor, and A. Bosio, "Investigating data representation for efficient and reliable convolutional neural networks," *Microprocessors and Microsystems*, vol. 86, p. 104 318, 2021.

[9] A. Lutsenko, *No Bullshit Converter (Nobuco)*, https://github.com/AlexanderLutsenko/nobuco, Accessed: 2025-04-16.

[10] A. Ruospo, G. Gavarini, C. de Sio, J. Guerrero, L. Sterpone, M. Sonza Reorda, E. Sanchez, R. Mariani, J. Aribido, and J. Athavale, "Assessing Convolutional Neural Networks Reliability through Statistical Fault Injections," in *Proc. Design, Automation & Test in Europe Conference & Exhibition*, 2023, pp. 1–6. DOI: 10.23919/DATE56975.2023.10136998.

[11] A. Ruospo, E. Sanchez, L. Matana Luza, L. Dilillo, M. Traiola, and A. Bosio, "A Survey on Deep Learning Resilience Assessment Methodologies," *Computer*, vol. 56, no. 2, pp. 57–66, 2023. DOI: 10.1109/MC.2022.3217841.