Benchmark Suite for Resilience Assessment of Deep Learning Models

Benchmark Suite for Resilience Assessment of Deep Learning Models

Cristiana Bolchini, *Senior Member, IEEE*, Alberto Bosio, *Member, IEEE*, Luca Cassano, *Senior Member, IEEE*, Antonio Miele, *Senior Member, IEEE*, Salvatore Pappalardo, Dario Passarello, Annachiara Ruospo, *Member, IEEE*, Ernesto Sanchez, *Senior Member, IEEE*, Matteo Sonza Reorda, *Fellow, IEEE*, and Vittorio Turco, *Student Member, IEEE*

***Abstract*—The reliability assessment of systems powered by artificial intelligence (AI) is becoming a crucial step prior to their deployment in safety and mission-critical systems. Recently, many efforts have been made to develop sophisticated techniques to evaluate and improve the resilience of AI models against the occurrence of random hardware faults. However, due to the intrinsic nature of such models, the comparison of the results obtained in state-of-the-art works is crucial, as reference models are missing. Moreover, their resilience is strongly influenced by the training process, the adopted framework and data representation, and so on. To enable a common ground for future research targeting convolutional neural networks (CNNs) resilience analysis/hardening, this work proposes a first benchmark suite of deep learning (DL) models commonly adopted in this context, providing the models, the training/test data, and the resilience-related information (fault list, coverage, etc.) that can be used as a baseline for fair comparison. To this end, this research identifies a set of axes that have an impact on the resilience and classifies some popular CNN models, in both PyTorch and TensorFlow. Some final considerations are drawn, showing the relevance of a benchmark suite tailored for the resilience context.**

***Index Terms*—Benchmark, deep learning, fault injection, machine learning, reliability.**

Received 23 October 2024; revised 8 March 2025 and 15 May 2025; accepted 19 May 2025. Date of publication 9 June 2025; date of current version 6 January 2026. This work was supported in part by the Future AI Research (FAIR); in part by the National Center for HPC, Big Data and Quantum Computing both through the European Union Next-Generation EU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR)—MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3—D.D. 1555 11/10/2022 under Grant PE00000013 and INVESTIMENTO 1.4—D.D. 3138 16/12/2021 mod. 3175 18/12/2021 under Grant CN00000013); and in part by the ANR France 2030 AdaptING Project under Grant "ANR-23-PEIA-0009." This article was recommended by Associate Editor G. Di Natale. *(Corresponding author: Antonio Miele.)*

Cristiana Bolchini, Luca Cassano, Antonio Miele, and Dario Passarello are with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, 20133 Milan, Italy (e-mail: cristiana.bolchini@polimi.it; luca.cassano@polimi.it; antonio.miele@polimi.it; dario.passarello@polimi.it).

Alberto Bosio and Salvatore Pappalardo are with Centrale Lyon, INSA Lyon, CNRS, Université Claude Bernard Lyon 1, CPE Lyon, INL, UMR5270, 69130 Ecully, France (e-mail alberto.bosio@ec-lyon.fr; salvatore.pappalardo@ec-lyon.fr).

Annachiara Ruospo, Ernesto Sanchez, Matteo Sonza Reorda, and Vittorio Turco are with the Department of Control and Computer Engineering, Politecnico di Torino, 10129 Turin, Italy (e-mail: annachiara.ruospo@polito.it; ernesto.sanchez@polito.it; matteo.sonzareorda@polito.it; vittorio.turco@polito.it).

Digital Object Identifier 10.1109/TCAD.2025.3578297

I. Introduction and Motivation

CURRENTLY, the adoption of artificial intelligence (AI) in safety-critical systems represents a significant advance that holds both promise and challenges. AI is currently employed in several areas, from autonomous vehicles and transportation, to aviation and aerospace systems. It is also widely used to assist in diagnosing medical conditions by interpreting medical images, and predicting patient outcomes. For their outstanding computational capability, AI algorithms, and deep learning (DL) in particular, can analyze vast amount of data (more than conventional machine learning algorithms [1]) and yield a prediction, e.g., make real-time decisions. By identifying patterns and anomalies, AI can predict potential failures and trigger maintenance actions before a critical failure occurs. However, the integration into *unacceptable-risk* and *high-risk* systems (as defined by the EU AI Act [2]) poses several concerns in terms of dependability guarantees. To make sure that these systems adhere to the highest safety standards, a rigorous approach to AI development, validation, and regulation is necessary.

In this context, academia and industry are putting a lot of effort in exploring reliability-related aspects in relation to AI and the last decade witnessed numerous studies addressing both the analysis and the hardening of such class of applications against hardware faults. A birds'-eye view of all these efforts and results is offered by these recent surveys [3], [4], [5].

However, when optimizing a convolutional neural network (CNN) to achieve the best performance from a purely AI perspective, it might suffice to specify the net, the training/test sets and adopted process, and the achieved accuracy. These elements allow for a useful and fair comparison to discriminate between alternative solutions. Yet, when introducing the hardware resilience focus, the specific elements of the implemented CNN are of paramount importance, since both analysis and hardening techniques actually exploit such elements to explore the fault/error relation and its impact on the final CNN accuracy when affected by a hardware fault.

Let us consider, as an example, five implementations of the same ResNet20 CNN, exploiting the PyTorch framework, trained on the same CIFAR-10 dataset and achieving a final comparable accuracy (∼91%). Given the five implementations, a software fault injection campaign has been executed injecting *the same list of faults* in the same fault locations (i.e.,

TABLE I
DIFFERENT IMPLEMENTATIONS OF THE SAME ARCHITECTURE (RESNET20) ON THE SAME DATASET (CIFAR10) TRAINED ON THE PYTORCH FRAMEWORK, LEADING TO DIFFERENT RESILIENCE FIGURES

DL model	Dataset	Training Epochs	Optimizer	Learning Decay	Final Golden Accuracy [%]	Number of Injected Faults (same fault list)	Masked [%]	Non-Critical [%]	SDC-1 [%]
ResNet20	CIFAR10	200	SGD	0,1 100/150	91.88	16,627	20.84	76.96	2.18
ResNet20	CIFAR10	200	SGD	0,2 60/120/180	91.91	16,627	20.24	77.55	2.20
ResNet20	CIFAR10	250	SGD	0,2 50/100/150/200	91.10	16,627	19.76	77.98	2.25
ResNet20	CIFAR10	405	adam	0,1 135/225/315	91.46	16,627	34.60	63.42	1.97
ResNet20	CIFAR10	250	SGD	0,2 50/100/150/200	91.83	16,627	21.69	76.07	2.22

CNN weights). Table I reports the results of the resilience assessment campaign. Faults' sample has been selected with a network-wise statistical fault injection [6] on a population of 17 174 144 stuck-at-faults with an error margin of 1% and a confidence level of 99%. Faults are classified as 1) *Masked* if they do not affect the final vector score at all; 2) *Noncritical* if they have an impact on the final vector score, but they keep the correct prediction; and 3) Silent Data Corruption (*SDC-1*) if they produce a wrong prediction, i.e., an application's failure [7]. As shown, the same faults lead to different effects based on the selected training implementation. Even though the difference of wrong predictions (also known in the literature as SDC-1) may seem negligible, a 0.28% difference (as observed between results in rows 3 and 4 in Table I) corresponds to 465 556 different predictions over 166 270 000 inferences (for each injected fault, the entire CIFAR10 testset was run, i.e., 10 000 images).

Indeed, the implementations are characterized by different weight distributions, an aspect that does impact the CNN performance [8], yet they could be considered similar given the similar accuracy. However, the two implementations yield two different solutions with respect to resilience to hardware-induced faults. Therefore, to be able to set baselines and compare alternative solutions when evaluating resilience and applying hardening strategies, it is necessary to have access to the same CNN implementation details. Indeed, the CNN implementation details, together with the adopted framework and the training dataset and process yield different characteristics from the hardware resilience point of view. Furthermore, accuracy per-se is of limited interest, whereas relative loss/improvement can offer insight, once all other aspects are known.

Based on the above considerations and given the dynamic and broad research community on the specific hardware resilience focus, we believe it is *timely and relevant* to define a benchmark suite of CNNs to be exploited when investigating hardware resilience (as also mentioned in the European Union on AI Act [9]), from both the analysis and the hardening points of view. To this end, this contribution is three-fold: 1) share commonly available implemented CNNs (and their implementation details) allowing the community to save time and efforts; 2) set a common ground for a sound and fair comparison between alternative approaches; and 3) move toward an ecosystem of innovative and alternative methods and tools, that gathers the many efforts of the research community.

The proposed suite identifies and exposes the elements that impact the application resilience in relation to hardware faults, as they emerge from the numerous studies in the field, such that every benchmark provides all the relevant details on the aspects that affect resilience-related metrics. At present, the set of benchmarks includes 16 CNN models trained on four popular image classification and segmentation datasets (i.e., CIFAR10, CIFAR100, GTSRB and COCO), being among the most commonly adopted ones according to the works analyzed in [3], [4], [5], and [10]. The benchmark suite is publicly available on GitHub [11].

The rest of the manuscript is organized as follows. The next section introduces the characterization of the adopted design choices, highlighting the aspects that affect resilience-related analyzes and metrics, as well as framework-related impacts. Section III describes the defined benchmark suite. It has been employed in an extensive reliability assessment campaign described in Section IV. Finally, Section V discusses related work and Section VI draws final conclusions.

II. CHARACTERIZATION ELEMENTS

When implementing a CNN, several elements have an impact not only on the performance of the application from a pure functionality standpoint, but also on the aspects that affect the system resilience against hardware faults. In this section, we identify such elements, and each benchmark will be characterized according to them. Although the set at present covers all elements of interest, this classification framework can be adapted and extended to introduce additional aspects.

1) *Machine Learning (ML) Framework:* The first and most important element that affects resilience is the adopted framework, because of the implementation internals that impact how data is stored and manipulated, and different computation methods yield different fault effects. As we observe the AI community move toward the adoption of ONNX [12] as an open-source standard to enable interoperability between different frameworks and hardware platforms. However, the main limitation when adopting ONNX is the lack of a direct access to the model internals (which is the asset of the framework) based on the fact that they are not relevant with respect to the expected functionality. While this is indeed the case when exploiting the model per second, the effects of faults in the hardware executing the application strongly depend on internals of the model and application being executed. Therefore, the only target for fault locations that would be consistent and homogeneous across different implementations of the same model would be faults in the weights and activation functions, a limited scope with respect to the suite here proposed. As the

experimental results will show, the ML framework has an impact on the internal organization of the network, such that, *as far as the resilience point of view is concerned*, any specific result pertains only the adopted framework and should not straightforwardly be applied to any other implementation. As such, the identified entries for this characterization elements are as follows.
   a) TensorFlow.
   b) PyTorch.
2) *Hardware Platform:* The platform where the application is executed affects the resilience analysis/hardening process with respect to both the adopted fault model, fault locations and the resulting error models (e.g., the two works in [13] and [14] perform the same error characterization analysis on two different architectures and obtain considerably different results). The identified entries are as follows.
   a) CPU.
   b) graphic processing unit (GPU).
   c) tensor processing unit (TPU).
   d) field programmable gate array (FPGA).
   e) Systolic array.
   f) ASIC.
3) *Task:* This element specifies the application class, useful to make sure CNNs taken from the same class are compared. The identified entries are as follows.
   a) Image classification.
   b) Object detection.
   c) Image segmentation.
   d) Regression.
4) *Dataset:* A fundamental aspect having an impact not only on the specific quality/performance of the final implementation but also on the resilience aspect. We identify as important parameters to be specified and made available, the following subelements.
   a) *Source:* The source of the dataset used to train the network and to feed it to evaluate resilience.
   b) *Type:* The type of data processed by the CNN.
   c) *Training/Test Sets:* This parameter specifies what portion of the dataset is used to train the CNN and which one to test it, to allow one to achieve a similar trained model, and to exercise it with the same set of input data when evaluating its resilience. Therefore, not only the cardinality of the sets is provided, but also the specific data.
5) *Training Process:* CNN resilience is greatly influenced by their training process. Important training parameters, including learning rate, batch size, number of epochs, dropout rate, weight regularization, and data augmentation, are crucial in determining how well a model remain resilient against random-hardware faults [15] and adversarial inputs [16].
6) *Data Representation:* As for the previous aspect, this element has an impact not only on the application quality/performance but also on its resilience. In recent years, various solutions have been proposed to tackle the growing complexity of DNNs in terms of memory usage and performance. One popular approach is pruning, which involves removing redundant neurons and weights from the networks. This results in sparse tensor parameters, thereby reducing computational costs and memory usage. Another common technique is the quantization [17] that modifies the data representation of DNN parameters (e.g., weights, activations) in order to reduce bit-width. One example is to convert from floating point 32-bits to integer 8-bits data representations. This results in lower memory footprint. A common aspect of both the pruning and quantization techniques is that they impact the overall DNN accuracy. To mitigate the accuracy reduction because of that, several works proposed pruning and quantization-aware training. Additionally to the quantization to integer data, new data formats with reduced bit widths, such as Brain Float [18] and Posit [19], [20], have been introduced and can be utilized during both CNN training and inference.
   Using a data representation with a reduced bit-width can significantly enhance the performance of a CNN during inference. However, it is crucial to consider the potential impact on the network's overall resilience. Numerous studies (e.g., [21], [22], [23], and [24]) investigate the effects of different data representations, therefore benchmarks are also characterized with respect to this parameter. The following entries are considered.
   a) *Floating Point:* FP8, FP16, FP32.
   b) *Integer:* INT8, INT16, INT32.
   c) Brain Float.
   d) *POSIT:* POSIT16, POSIT32.

III. Proposed Benchmark Suite

The main intent of the proposed research work is to define a benchmark suite based on the characterization elements described in the previous section. Indeed, all identified parameters would generate a huge set of possible combinations; therefore, in this first work, we restricted our view by filtering on a subset of them, thus obtaining a first proposal. Our aim is to provide a public reference point and later to involve the community in the future to extend the benchmark suite based on the specific research interests. The current version of the benchmark suite is freely downloadable from [11].

As reported in Table II, the suite contains 11 different CNN architectures performing image classification and segmentation, trained and tested on popular dataset, for a total of 16 CNN models. We selected these specific tasks since they are the most commonly considered by the referred research community. The selected CNN architectures are: ResNet18, ResNet20, ResNet32, and ResNet44 [25], [26], GoogLeNet [27], DenseNet121 and DenseNet161 [28], MobileNetV2 [29], VGG-11 and VGG-13 [30], and DeepLabV3 [31]. As mentioned, the CNN architectures have been trained on well popular datasets, namely, CIFAR10 and CIFAR100 [32], GTSRB [33], and COCO [34]. Then, they have been validated on the same datasets, except for DeepLabV3 which has been trained on COCO (with PASCAL VOC classes), and validated on PASCAL VOC 2012 [35] validation dataset).

TABLE II
AVAILABLE CNN MODELS IN THE PROPOSED BENCHMARK SUITE. THE LAST COLUMN (I.E., ACCURACY METRIC [%]) IS A FAULT-FREE TASK-SPECIFIC MEASURE: IT REPORTS THE † TOP-1 ACCURACY IN IMAGE CLASSIFICATION MODELS, AND THE § MEAN PIXEL ACCURACY (MPA) AND MEAN INTERSECTION OVER UNION (MIOU) IN THE IMAGE SEGMENTATION BENCHMARK, RESPECTIVELY

ML Framework	HW Platform	Task	Dataset	Data Representation	DL Model	Accuracy Metric [%]
PyTorch	GPU	Image Classification†	CIFAR10	FP32	DenseNet121	93.16
					DenseNet161	93.06
					GoogLeNet	92.17
					MobileNetV2	91.72
					ResNet20	91.47
					ResNet32	92.34
					ResNet44	92.75
					VGG-11	91.31
					VGG-13	93.76
			CIFAR100	FP32	DenseNet121	78.73
					GoogLeNet	76.28
					ResNet18	76.18
			GTSRB	FP32	DenseNet121	96.54
					ResNet20	94.28
					VGG-11	95.52
		Image Segmentation§	PASCAL VOC	FP32	DeepLabV3	89.18, 71.06
TensorFlow	GPU	Image Classification†	CIFAR10	FP32	DenseNet121	93.16
					DenseNet161	93.06
					GoogLeNet	92.17
					MobileNetV2	91.72
					ResNet20	91.47
					ResNet32	92.34
					ResNet44	92.75
					VGG-11	91.31
					VGG-13	93.76
			CIFAR100	FP32	DenseNet121	78.73
					GoogLeNet	76.28
					ResNet18	76.18
			GTSRB	FP32	DenseNet121	96.54
					ResNet20	94.28
					VGG-11	95.52
		Image Segmentation§	PASCAL VOC	FP32	DeepLabV3	89.2, 71.08

The CIFAR-10 dataset consists of 60 000 32×32 color images in 10 classes, with 6000 images per class. There are 50 000 training images and 10 000 test images [36]. CIFAR-100 is close to CIFAR-10, but it has 100 classes containing 600 images each. For each class there are 500 training images and 100 test images. The 100 classes of CIFAR-100 are grouped into 20 superclasses. Each image has a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs). GTSRB contains more than 50 000 images of German traffic signs, categorized into 43 classes. Regarding the segmentation dataset, PASCAL VOC 2012 contains over 11 530 images with featuring objects from 20 classes. It provides annotations for object detection, segmentation, and classification, with standard splits for training, validation, and testing.

Among the available ML frameworks, this work supports both PyTorch and TensorFlow. Indeed, one challenge has been to define equivalent models (i.e., having same structure, hyper-parameters and parameters) in both frameworks. Therefore, we carried out the design and training of each CNN in PyTorch and later adopted a semi-automated converter to obtain the corresponding TensorFlow model.

PyTorch models have been downloaded from [37], [38], and [39]. Then, each of the selected models was adapted and modified to be matched and trained with multiple datasets, thus obtaining 16 benchmarks. Tables III and IV report all the design and training information for each model/dataset pair that we have selected, respectively. It is worth noting that the number of parameters in a model can vary when applied to different datasets. This variation is largely due to differences in the internal architecture tailored to each dataset, particularly the final layer of the model. The final layer is typically customized to match the number of classes in each dataset: for instance, 10 classes for CIFAR-10, 100 classes for CIFAR-100, and 43 classes for GTSRB. In this research work, the data type in the selected models correspond to the IEEE Standard for Floating-Point Arithmetic (IEEE 754),

TABLE III
SELECTED CNNS: ARCHITECTURAL DESCRIPTIONS

CNN Architecture	Layer Types
ResNet18	[Conv2d(17), Linear(1)]
ResNet20	[Conv2d(19), Linear(1)]
ResNet32	[Conv2d(31), Linear(1)]
ResNet44	[Conv2d(43), Linear(1)]
DenseNet121	[Conv2d(120), Linear(1)]
DenseNet161	[Conv2d(160), Linear(1)]
GoogLeNet	[Conv2d(66), Linear(1)]
MobileNetV2	[Conv2d(57), Linear(1)]
VGG-11	[Conv2d(8), Linear(3)]
VGG-13	[Conv2d(10), Linear(3)]
DeepLabV3	[Conv2d(63)]

TABLE IV
CNNs TRAINING INFORMATION. NOTES: LOSS FUNCTION IS CROSSENTROPYLOSS; COCO IS RESTRICTED TO ONLY PASCAL VOC CLASSES; § POLYNOMIAL

Model	Dataset	Params	Optimizer	Batch	Epochs	Learning Rate (LR)	LR decay	Milestones
ResNet20	CIFAR10	269,722	Adam	128	200	0.001	0.1	[50, 100, 150]
ResNet32	CIFAR10	464,154	Adam	128	200	0.001	0.1	[50, 100, 150]
ResNet44	CIFAR10	658,586	Adam	128	200	0.001	0.1	[50, 100, 150]
GoogLeNet	CIFAR10	5.49 M	Adam	128	250	0.001	0.2	[50, 100, 150,200]
DenseNet121	CIFAR10	6.97 M	Adam	128	250	0.001	0.2	[50, 100, 150,200]
DenseNet161	CIFAR10	26.49 M	Adam	128	250	0.001	0.2	[50, 100, 150,200]
MobileNetV2	CIFAR10	2.3 M	Adam	128	250	0.001	0.2	[50, 100, 150,200]
VGG-11	CIFAR10	28.15 M	Adam	128	250	0.001	0.2	[50, 100, 150,200]
VGG-13	CIFAR10	28.33 M	Adam	128	250	0.001	0.2	[50, 100, 150,200]
ResNet18	CIFAR100	11.2 M	SGD	128	200	0.1	0.02	[60, 120, 160]
GoogLeNet	CIFAR100	6.2 M	SGD	128	200	0.1	0.02	[60, 120, 160]
DenseNet121	CIFAR100	7 M	SGD	128	200	0.1	0.02	[60, 120, 160]
ResNet20	GTSRB	271,867	Adam	128	200	0.001	0.1	[50, 100, 150]
VGG-11	GTSRB	28.29 M	Adam	128	200	0.001	0.1	[50, 100, 150]
DenseNet121	GTSRB	6.99 M	Adam	128	200	0.001	0.1	[50, 100, 150]
DeepLabV3	COCO	42 M	SGD	32	30	0.02	0.9 §	-

single precision 32-bit floating point data (FP32). Indeed, the adopted data representation has also an impact on reliability-related aspects, therefore we envision that in the future the suite will include alternative versions of the same networks implemented with different data representations/quantization.

We have automated the translation of PyTorch models CNNs in the equivalent TensorFlow counterparts by means of an open-source Python tool called Nobuco [40]. The tool extracts the graph together with hyperparameters, parameters and weights related to each included operator. Moreover, it requires some manual adjustment to define 1) the input and output tensor format (e.g., from BCHW to BHWC) and 2) the mapping between PyTorch and TensorFlow operators (for instance, the tool does not automatically maps softmax operations). The converter is invoked by means of a Python script, where required customization actions are specified. This conversion (from PyTorch to TensorFlow) allowed us to obtain two CNN models having (almost) the *same* golden accuracy, as shown in Table II, Column 7. Even employing all these efforts in the conversion of the models, some minor differences can be observed among models and tools. Given the experiments we performed, we believe any mismatch in the software stack could have contributed to the observed differences. For example, different drivers for the GPU could have slightly different implementations and result in different values, even when using the same tools.

The training process and the subsequent reliability assessments have been executed targeting the CNN software models, as it will be deepened in Section IV, but all the experiments have been run on the popularly used GPU devices, i.e., the experiments on PyTorch were conducted on hardware comprising an Intel Xeon Gold 6238R CPU @ 2.20GHz equipped with an NVIDIA GeForce RTX 3060 Ti GPU with 8 GB of Memory. Most of the experiments performed on TensorFlow have been run on a 13th Gen Intel Core i7-13700K equipped with a GPU NVIDIA RTX 4060t with 16GB of DDR6 memory.

DeepLabV3 experiments were performed on a Intel Xeon Silver 4210 CPU @ 2.20 GHz equipped with an NVIDIA Tesla V100 GPU with 32 GB of Memory.

The defined benchmark suite has been completed with fault lists and results of the related experimental runs for reliability analysis purposes. These aspects are discussed in the next section. The above reported information, available in the public repository, annotates the adopted complete process to characterize each benchmark model and support experimental reproducibility.

IV. RELIABILITY ASSESSMENT

To measure the resilience of the released benchmarks against the occurrence of random-hardware faults, we have performed statistical fault injections (SFIs) targeting faults affecting the synaptic weights (i.e., the implementation details) of the considered benchmark models based on the methodology proposed in [6]. For each model, an SFI process was conducted to obtain a network-wise estimate of the resilience with an error margin between 1% and 3% and a 99% confidence that the exhaustive result (unknown) fall within the statistical range. Experiments were performed on the developed benchmark models for image classification and segmentation.

Fault injection campaigns were performed on both PyTorch models and their converted version in TensorFlow. As for the PyTorch models, a PyTorch-based fault injector, SFIadvancedmodels [41], was used, which is capable of simulating software-level faults on both the parameters and neural layers of the models. We implemented a TensorFlow conterpart of SFIadvancedmodels, freely downloadable from [42]. Stuck-at faults were injected into all the convolutional and linear layers of the models, the considered fault model is the permanent fault (stuck-at-0/1) on a synaptic weight acting on a single bit of the 32-bit floating point (FP32) weights. Fault lists were generated for PyTorch models; then, since same operators have different naming conventions in the two considered ML frameworks, we have defined a Python script performing a find/replace on the fault list to translate it from PyTorch to TensorFlow.

The total number of injected faults and the number of inputs used for analysis are reported in Table V for all the considered

TABLE V
FI DETAILS FOR A NETWORK-WISE STATISTICAL ESTIMATE WITH A 99% CONFIDENCE AND A 1% MARGIN OF ERROR FOR CIFAR-10, CIFAR-100, GTRSB, AND 3% FOR PASCAL VOC 2012

Dataset	Model	Injected Faults	Test Set Images
CIFAR10	ResNet20	16,627	10,000
	ResNet32	16,630	
	ResNet44	16,634	
	DenseNet121	16,628	
	DenseNet161	16,677	
	GoogLeNet	16,640	
	MobileNetV2	16,640	
	VGG-11	16,642	
	VGG-13	16,639	
CIFAR100	ResNet18	16,644	10,000
	GoogLeNet	16,635	
	DenseNet121	16,638	
GTRSB	ResNet20	16,629	12,600
	VGG-11	16,641	
	DenseNet121	16,650	
PASCAL VOC	DeepLabV3	1,852	1,449

datasets. For the sake of reproducibility, all the fault lists have been released in our public repository, for both the PyTorch and the TensorFlow frameworks.

A. Experimental Campaigns Duration

To set up and perform all the experiments reported in these campaigns to provide the benchmark suite in its present status, an overall of continuous 38 days have been necessary (an estimation of continuous machine/computational time), running them on the processing platforms presented in Section III.

B. Discussion and Final Remarks

The novel contribution of this article stems from the lack of any other (large enough) set of models analyzed and reported in the same consistent and complete fashion. The benchmark models here proposed constitute a broad consistent set of elements to be used for further developments. The implemented networks are the nominal ones, not aimed at being robust or exhibiting particular resilience against hardware faults, thus the reported discussion focuses on highlighting implementation implications on resilience. More precisely, the results in Table I preliminary introduce how different implementations and trainings of the same network affect the system's resilience. It is observed that resilience analyzes of the same model can vary significantly. For example, training with the Adam optimizer masks at least 12% more injected faults and simultaneously exhibits better SDC-1 fault mitigation capabilities. In addition, another obstacle is the framework used for training. Identical architectural configurations on different frameworks can yield varying training results. Indeed, previous studies, such as [43] and [44], compare different tools, highlighting how the algorithms and libraries employed directly influence network performance. Thus, even with consistent hyperparameters, different frameworks can produce neural networks that differ in weight values and accuracy. Therefore, to ensure a fair comparison between different solutions—whether for analysis or resilience improvement—a shared set of benchmarks with consistent weights across different frameworks is essential.

In our FI experiments, detailed in Tables VI, VII, VIII, and IX, a conversion of PyTorch models and their weights was done to obtain equivalent TensorFlow versions. The primary objective was to achieve identical accuracy between the two frameworks. This goal was met, as evidenced in reported results (Tables VI, VII, VIII, and IX) where the second column shows the Golden Accuracy metric shared across all models and datasets. This consistency indicates no significant variation in accuracy between the PyTorch and TensorFlow implementations, underscoring the accuracy of the conversion process.

The second phase of the experiments involved assessing the resilience of both PyTorch and converted TensorFlow models through fault injection campaigns. Both models were subjected to identical fault conditions, the same fault list is used to introduce permanent faults into the synaptic weights of the neural networks. This approach ensures a fair comparison of the models' responses to the faults. From an initial analysis, most models across all three image classification datasets show slight differences between masked and noncritical inferences. In particular, all converted TensorFlow ResNet models exhibit superior masking abilities compared to the PyTorch versions across all three datasets. The smallest difference is 4.09% with the ResNet18 network trained on CIFAR100, while the largest difference is seen in the ResNet20 network trained on the GTSRB dataset, which achieves a discrepancy of 6.82%. Conversely, MobileNetV2 behaves in the opposite way, with its PyTorch version having more than 15% masked inferences compared to its TensorFlow counterpart. It is important to emphasize that these percentage differences between masked and nonmasked are mostly absorbed by noncritical inferences, and vice versa. Among the models mentioned so far, the difference in the most critical inferences, those where SDC-1 faults are injected, is minimal. The only notable variation is between the ResNet18 (GTSRB) models, with a gap of 0.280%, while all others range are between 0.000% and 0.006%.

For the DenseNet and VGG models, the impact of the fault injection campaign across the two frameworks is quite similar. Variations in masked and noncritical inferences are minimal, with only the DenseNet121 model trained on CIFAR100 showing differences exceeding 1%, compared to the models trained on the other two datasets. In terms of SDC-1 inferences, the discrepancies are also minimal, with the largest differences observed in networks trained on CIFAR10, ranging from 0.049% in VGG-11 to 0.074% in DenseNet121, while for other datasets, these differences do not exceed 0.002%.

In contrast, the GoogLeNet model shows significant variations depending on the dataset used for training. While the difference in SDC-1 inferences between the two frameworks is negligible, the TensorFlow version exhibits 6.12% more noncritical inferences, compensated by masked inferences, when trained on the GTSRB dataset. However, with the CIFAR10 dataset, it shows a reduction of 0.24% in noncritical inferences.

TABLE VI
CIFAR10 RESILIENCE STATISTICS

Model	Golden Top-1 Accuracy [%]	Faulty Outputs	PyTorch				TensorFlow			
			Masked [%]	Non-Critical [%]	SDC-1 [%]	Faulty Top-1 Accuracy [%]	Masked [%]	Non-Critical [%]	SDC-1 [%]	Faulty Top-1 Accuracy [%]
ResNet20	91.47	166,270,000	41.65	57.08	1.28	90.35	46.65	52.08	1.27	90.37
ResNet32	92.34	166,300,000	43.08	55.48	1.44	91.04	48.61	49.95	1.44	91.05
ResNet44	92.75	166,340,000	42.39	56.32	1.29	91.58	48.12	50.59	1.29	91.56
DenseNet121	93.16	166,280,000	83.04	15.89	1.07	92.18	82.17	16.84	1.00	92.23
DenseNet161	93.06	166,770,000	89.67	9.22	1.10	92.05	89.43	9.46	1.10	92.30
GoogLeNet	92.17	166,400,000	85.49	13.31	1.20	91.09	83.67	15.13	1.21	91.10
MobileNetV2	91.72	166,400,000	65.09	32.42	2.49	89.47	49.35	48.16	2.49	89.48
VGG-11	91.31	166,420,000	78.62	20.24	1.15	90.29	78.90	20.00	1.10	90.35
VGG-13	93.76	166,390,000	78.67	20.17	1.15	92.76	79.00	19.91	1.09	92.76

TABLE VII
CIFAR100 RESILIENCE STATISTICS

Model	Golden Top-1 Accuracy [%]	Faulty Outputs	PyTorch				TensorFlow			
			Masked [%]	Non-Critical [%]	SDC-1 [%]	Faulty Top-1 Accuracy [%]	Masked [%]	Non-Critical [%]	SDC-1 [%]	Faulty Top-1 Accuracy [%]
ResNet18	76.18	166,440,000	74.39	24.32	1.28	75.22	78.48	20.52	1.00	75.43
GoogLeNet	76.28	166,350,000	69.33	29.59	1.09	75.47	63.20	35.71	1.09	75.47
Densenet121	78.73	166,380,000	64.82	33.43	1.75	77.37	63.62	34.64	1.75	77.38

TABLE VIII
GTSRB RESILIENCE STATISTICS

DL model	Golden top-1 Accuracy[%]	Faulty Outputs	PyTorch				TensorFlow			
			Masked [%]	Non-Critical [%]	SDC-1 [%]	Faulty Top-1 Accuracy [%]	Masked [%]	Non-Critical [%]	SDC-1 [%]	Faulty Top-1 Accuracy [%]
ResNet20	94.28	209,999,010	31.20	67.32	1.49	92.86	38.02	60.50	1.50	92.91
VGG-11	95.52	210,188,460	89.80	9.76	0.45	95.10	89.79	9.76	0.44	95.10
DenseNet121	96.54	210,289,500	40.18	57.87	1.95	94.66	40.95	57.10	1.95	94.67

TABLE IX
PASCAL VOC 2012 RESILIENCE STATISTICS. (A) ACCURACY METRICS ANALYSIS. (B) INFERENCES ANALYSIS.

(a)

DL model	PyTorch				TensorFlow			
	Golden Pixel Accuracy[%]	Golden mIoU[%]	Faulty Pixel Accuracy[%]	Faulty mIoU[%]	Golden Pixel Accuracy[%]	Golden mIoU[%]	Faulty Pixel Accuracy[%]	Faulty mIoU[%]
DeepLabV3	89.18	71.06	88.85	55.85	89.20	71.08	89.01	54.50

(b)

DL model	PyTorch			TensorFlow		
	Masked SDC [%]	Tolerable SDC [%]	Critical SDC [%]	Masked SDC [%]	Tolerable SDC [%]	Critical SDC [%]
DeepLabV3	79.90	18.75	1.35	88.69	9.00	2.31

Finally, Table IX presents the analysis of the DeepLabV3 image segmentation model with the PASCAL VOC 2012 validation dataset. Unlike image classification, the approach to categorize faulty inferences in this task differs because segmentation produces multiple predictions per image. Following the methodology outlined in [10], we define faults as follows.

1) *Masked SDC:* The predicted pixel labels in the faulty and golden outputs are identical.
2) *Tolerable SDC:* Less than 1% of pixels are modified between the faulty and golden outputs, with no new classes appearing or existing classes disappearing.
3) *Critical SDC:* At least 1% of pixels are modified between the faulty and golden outputs, or new classes appear/disappear.

Regarding the results, Table IX shows that the conversion from PyTorch to TensorFlow yielded similar outcomes in terms of average faulty pixel accuracy and IoU between the two frameworks. However, as seen in Table IX, the model exhibits significant differences across all three classification categories. Notably, the number of critical inferences is nearly 1% higher in TensorFlow compared to PyTorch. We inspected this behavior and we argue that the non-negligible differences between the two ML-frameworks are due to the sensitivity of the two convolutional algorithms (e.g., Winograd, GEMM), which cannot be forced.

Experimental results show that despite starting from a model with the same accuracy and internal data (weights), the framework (e.g., PyTorch versus TensorFlow) and the internal tool representation play an important role in fault propagation, due to different types of rounding, for example. The problem of *numerical inaccuracies* is not new in the literature: it is highly discussed in [45] where the numerical inaccuracy

problem is addressed for mainstream DNN hardware implementations with floating-point and fixed-point data types. The round-off errors in successive multiply and accumulate (MAC) operations vary depending on the chosen data type and the machine learning framework, resulting in different patterns of fault propagation.

V. Related Work

When looking at dependability related matters, both analysis and hardening strategies often need to access the specific implementation details of the system being investigated, to examine the fault-error relation, at various levels of abstraction. This kind of information is typically not of interest to the ML and DL community. In fact, benchmarks for AI systems aim to 1) evaluate the performance of the hardware itself (e.g., the CPUs, GPUs, TPUs); 2) assess various aspects of the ML model, including metrics like accuracy as well as system resource utilization; and 3) address various aspects of the data used to train and test ML models, including quality, diversity and bias. Examples are [46], micro-benchmarks, a collection of basic operations (dense matrix multiplies, convolutions and communication) as well as some recurrent layer types that can be executed on an arbitrary framework/hardware platforms. Reference [47] is a part of the MLCommons project (https://mlcommons.org/benchmarks/), an industry standard suite of macro-benchmarks. Among the latter, there is also a specific *AI Safety Benchmarks* suite, focused on enabling the assessment of the safety for a particular use case (e.g., application, user personas) by enumerating hazards and then testing a system for appropriate handling of prompts that could enable those hazards. Indeed the aim of the proposed suite is different and the safety implied by the application can actually be explored on top of the resilience of the AI application to hardware faults. There are also other ML benchmarks, possibly focused on a well-defined area of interest and only providing a limited number of networks. For example, the EEMBC MLMark Benchmark (https://arc.net/l/quote/sbhrtjyj) presents a system to measure the performance and accuracy of embedded inference. Further examples of open-source machine learning and deep learning benchmark suites are [48], or the ML library focused on edge applications of the EDA group at Politecnico di Torino https://github.com/eml-eda/pytorch-benchmarks/tree/main.

Furthermore, considering domain-specific datasets, it is worth mentioning OBPMark (On-Board Processing Benchmarks), a set of representative open-source benchmarks for space applications, developed in cooperation by the European Space Agency and Barcelona Supercomputing Center [49]. The work aims at benchmarking standard on-board processing functions, to compare end-user performance of different devices and systems.

As far as the availability of ML applications and implementations used for resilience analysis/hardening, all work carried out by the Dependable System Laboratory at the University of British Columbia (https ://github.com/orgs/DependableSystemsLab/ repositories) releases all tools and the networks, so that fair and sound baselines can be set. We believe it is timely to collaborate on systematically collecting well-annotated models along with pertinent information to establish a shared reference enabling comparison and improvement.

VI. Conclusion

This article presents a benchmark suite for resilience assessment and improvement in DL models. We initially focused on popular CNNs models for image classification and segmentation. We have identified a set of factors that have an impact on the reliability assessment of CNN models (e.g., ML framework, data representation, hardware platform). Based on that, we have provided a first set of benchmarks covering some axes and attributes. More specifically, a total of 11 CNN architectures have been implemented and analyzed, on four popular datasets (CIFAR-10, CIFAR-100, GTRSB, and PASCAL VOC 2012). When considering two different implementations of the same CNN model exhibiting the same accuracy and implemented using the same internal data (e.g., weights) deviate in a similar way when injecting faults (e.g., perturbing) in the weights, that is the memory storing such data. However, if faults are injected in the processor executing the CNN layer, the effects on the layer output (and later on the CNN output) strictly depend on the layer internal implementation, that may be different. As a result, the two models yield different resilience characteristics. Therefore, when comparing different approaches, the same implementation has to be adopted for a fair reference, making the availability of benchmarks of paramount importance. As mentioned, because of the need to access the details of the implemented models to perform a comparable resilience related analysis, ONNX has not been considered as a viable framework since it makes available only a subset of all layers, thus not enabling at present a complete access.

The present suite is an initial contribution, as we expect to expand it by, including models that cover other cases along the characterization axes, to offer a broader baseline support for the research community.

Acknowledgment

This article reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

References

[1] C. C. Aggarwal, *Neural Networks and Deep Learning*. Cham, Switzerland: Springer, 2023. [Online]. Available: https://link.springer.com/book/10.1007/978-3-031-29642-0

[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." Accessed: Apr. 16, 2025. [Online]. Available: https://eur-lex.europa.eu/resource.html?uri=cellar:e0649735-a372-11eb-9585-01aa75ed71a1.0001.02/DOC_1&format=PDF

[3] A. Ruospo, E. Sanchez, L. M. 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.

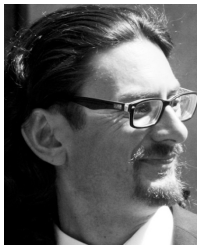
[4] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Comput. Surv.*, vol. 56, no. 6, pp. 1–39, 2024.

[5] C. Bolchini, L. Cassano, and A. Miele, "Resilience of deep learning applications: A systematic literature review of analysis and hardening techniques," *Comput. Sci. Rev.*, vol. 54, pp. 1–32, Nov. 2024.
[6] A. Ruospo et al., "Assessing convolutional neural networks reliability through statistical fault injections," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, 2023, pp. 1–6.
[7] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2017, pp. 1–12.
[8] E. Ozen and A. Orailoglu, "SNR: Squeezing numerical range defuses bit error vulnerability surface in deep neural networks," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5s, 1–25, 2021.
[9] "European union artificial intelligent act—Article 15: Accuracy, robustness and cybersecurity." Accessed: Apr. 16, 2025. [Online]. Available: https://artificialintelligenceact.eu/article/15/#
[10] S. Burel, A. Evans, and L. Anghel, "Techniques for detecting and masking faults in semantic segmentation applications," *Microelectron. Rel.*, vol. 157, Jun. 2024, Art. no. 115397. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271424000775
[11] "Deep neural network models for reliability studies." Accessed: May 7, 2025. [Online]. Available: https://github.com/ReADLBench/dnn-benchmark
[12] "ONNX." Accessed: Apr. 16, 2025. [Online]. Available: https://onnx.ai/
[13] C. Bolchini, L. Cassano, A. Miele, A. Nazzari, and D. Passarello, "Analyzing the reliability of alternative convolution implementations for deep learning applications," in *Proc. Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst.*, 2023, pp. 1–6.
[14] A. Veronesi et al., "Cross-layer reliability analysis of NVDLA accelerators: Exploring the configuration space," in *Proc. Int. Eur. Test Symp.*, 2024, pp. 1–6.
[15] U. Zahid, G. Gambardella, N. J. Fraser, M. Blott, and K. Vissers, "FAT: Training neural networks for reliable inference under hardware faults," in *Proc. IEEE Int. Test Conf. (ITC)*, 2020, pp. 1–10.
[16] C. Eleftheriadis, A. Symeonidis, and P. Katsaros, "Adversarial robustness improvement for deep neural networks," *Mach. Vis. Appl.*, vol. 35, p. 35, Mar. 2024.
[17] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, 1–14.
[18] S.-Q. Tong et al., "Energy-efficient brain floating point convolutional neural network using memristors," *IEEE Trans. Electron Devices*, vol. 71, no. 5, pp. 3293–3300, May 2024.
[19] J. L. Gustafson and I. T. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomput. Front. Innov.*, vol. 4, no. 2, pp. 71–86, 2017.
[20] R. Murillo, A. A. Del Barrio, and G. Botella, "Deep PeNSieve: A deep learning framework based on the posit number system," *Digit. Signal Process.*, vol. 102, Jul. 2020, Art. no. 102762.
[21] M. A. Neggaz, I. Alouani, S. Niar, and F. Kurdahi, "Are CNNs reliable enough for critical applications? An exploratory study," *IEEE Design Test*, vol. 37, no. 2, pp. 76–83, Apr. 2020.
[22] B. Reagen et al., "Ares: A framework for quantifying the resilience of deep neural networks," in *Proc. Design Autom. Conf.*, 2018, pp. 1–6.
[23] A. Ruospo, E. Sanchez, M. Traiola, I. O'Connor, and A. Bosio, "Investigating data representation for efficient and reliable convolutional neural networks," *Microprocess. Microsyst.*, vol. 86, Oct. 2021, Art. no. 104318.
[24] I. Alouani, A. B. Khalifa, F. Merchant, and R. Leupers, "An investigation on inherent robustness of posit data representation," in *Proc. Int. Conf. VLSI Design Int. Conf. Embed. Syst.*, 2021, pp. 276–281.
[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[26] S. Targ, D. Almeida, and K. Lyman, "Resnet in resnet: Generalizing residual architectures," 2016, *arXiv:1603.08029*.
[27] C. Szegedy et al., "Going deeper with convolutions," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[28] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4700–4708.
[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015, *arXiv:1409.1556*.
[31] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking Atrous convolution for semantic image segmentation," 2017, *arXiv:1706.05587*.
[32] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009. Accessed: Apr. 16, 2025. [Online]. Available: https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf
[33] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The German traffic sign recognition benchmark: A multi-class classification competition," in *Proc. Int. Conf. Neural Netw.*, 2011, pp. 1453–1460.
[34] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," 2015, *arXiv:1405.0312*.
[35] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. "The PASCAL visual object classes challenge 2012 (VOC2012) results." Accessed: May 7, 2025. [Online]. Available: http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.
[36] "CIFAR-10." Accessed: Apr. 16, 2025. [Online]. Available: https://www.cs.toronto.edu/ kriz/cifar.html
[37] Y. Idelbayev. "Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch." Accessed: Apr. 16, 2025. [Online]. Available: https://github.com/akamaster/pytorch_resnet_cifar10
[38] "PyTorch-cifar100." Weiaicunzai. Accessed: Apr. 16, 2025. [Online]. Available: https://github.com/weiaicunzai/pytorch-cifar100
[39] "DeepLabV3 model with ResNet50 backbone." Accessed: May 7, 2025. [Online]. Available: https://pytorch.org/vision/stable/models/generated/torchvision.models.segmentation.deeplabv3_resnet50.html
[40] A. Lutsenko. "No bullshit converter (Nobuco)." Accessed: Apr. 16, 2025. [Online]. Available: https://github.com/AlexanderLutsenko/nobuco
[41] A. Ruospo and V. Turco. "SFIadvancedmodels." Accessed: Apr. 16, 2025. [Online]. Available: https://github.com/cad-polito-it/SFIadvancedmodels
[42] D. Passarello, L. Cassano, A. Miele, and C. Bolchini. "keras_weight_injector." Accessed: Apr. 16, 2025. [Online]. Available: https://github.com/D4De/keras_weight_injector
[43] H. Kim, H. Nam, W. Jung, and J. Lee, "Performance analysis of CNN frameworks for GPUs," in *Proc. Int. Symp. Perform. Anal. Syst. Softw.*, 2017, pp. 55–64.
[44] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative study of deep learning software frameworks," 2016, *arXiv:1511.06435*.
[45] E. Ozen and A. Orailoglu, "Low-cost error detection in deep neural network accelerators with linear algorithmic checksums," *J. Electron. Testing*, vol. 36, no. 6, pp. 703–718, 2020.
[46] "DeepBench." Accessed: Apr. 16, 2025. [Online]. Available: https://github.com/baidu-research/DeepBench
[47] "MLPerf." MLPerf Inference Benchmarks. Accessed: Apr. 16, 2025. [Online]. Available: https://github.com/mlcommons/inference
[48] C. Banbury et al., "MLPerf tiny benchmark," in *Proc. Neural Inf. Process. Syst. Track Datasets Benchmarks*, 2021, 1–15.
[49] "OBPMark (on-board processing benchmarks)." Accessed: Feb. 2025. [Online]. Available: https://github.com/OBPMark/

Cristiana Bolchini (Senior Member, IEEE) received the Ph.D. degree in automation and computer engineering from the Politecnico di Milano, Milan, Italy, in 1997.

She is a Professor with the Politecnico di Milano. She has co-authored more than 150 papers in peer-reviewed international journals and conferences. Her research interests cover the areas of methodologies for the design and analysis of digital systems with a specific focus on dependability and self-awareness for heterogeneous system architectures.

Prof. Bolchini served as the Technical Program Chair of the conference IEEE/ACM Design Automation and Test in Europe.



Alberto Bosio (Member, IEEE) received the Ph.D. degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 2006.

He is currently a Full Professor with the École Centrale de Lyon, Institute of Nanotechnology, Lyon, France. He published articles spanning diverse disciplines, including testing, reliability, in-memory computing, approximate computing, and emerging technologies. Website: http://perso.ec-lyon.fr/alberto.bosio/.

Luca Cassano (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer engineering from the University of Pisa, Pisa, Italy, in 2006, 2009, and 2013, respectively.

He is an Associate Professor with the Politecnico di Milano, Milan, Italy. He co-authored more than 70 peer-reviewed scientific papers. His research activity focuses on fault simulation, testing, untestability analysis, diagnosis, and verification of fault tolerant, and secure digital circuits and systems.

Prof. Cassano has also been an Associate Member of the technical staff at Maxim Integrated.

Antonio Miele (Senior Member, IEEE) received the M.Sc. degree in computer engineering from the Politecnico di Milano, Milan, Italy, in 2006, and the second M.Sc. degree in computer science from the University of Illinois at Chicago, Chicago, IL, USA, in 2006, and the Ph.D. degree in information technology from the Politecnico di Milano, in 2010.

He is an Associate Professor with the Politecnico di Milano. He co-authored more than 90 peer-reviewed publications. His main research interests cover the design of fault tolerant computing systems and runtime resource management in heterogeneous multicores.

Salvatore Pappalardo received the master's degree in computer engineering from the Politecnico di Torino, Turin, Italy, in October 2022. He is currently pursuing the Ph.D. degree with the Ecole Centrale de Lyon, Écully, France.

His main research interests are reliable hardware design, approximate computing, and neural network in embedded systems.

Dario Passarello received the M.Sc. degree in computer science and engineering from the Politecnico di Milano, Milan, Italy, in 2023.

He formerly held a position as a Research Fellow with DEIB department, Politecnico di Milano. His work focused on studying the reliability of DNNs.

Annachiara Ruospo received the M.Sc. degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 2018.

She is currently an Assistant Professor with the Department of Control and Computer Engineering, Politecnico di Torino. Her main research interests include safety, reliability, and security aspects of AI systems, with a focus on artificial neural networks, and test and verification of modern embedded devices.

Dr. Ruospo is a member of the AI Existential Safety Community of the Future of Life Institute.

Ernesto Sanchez received the Ph.D. degree in computing engineer from the Politecnico di Torino, Turin, Italy, in 2006.

He is an Associate Professor with the Politecnico di Torino. His research interests include digital circuits and systems reliability, evolutionary computation, and ANN reliability.

Matteo Sonza Reorda (Fellow, IEEE) received the first M.Sc. degree in electronics and the Ph.D. degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 1986 and 1990, respectively.

He is currently a Full Professor with the Department of Control and Computer Engineering, Politecnico di Torino. He published more than 400 papers in the area of test and fault-tolerant design of reliable circuits and systems. He is involved in numerous research projects with companies and other research centers worldwide.

Prof. Sonza Reorda received several best paper awards at major international conferences.

Vittorio Turco received the master's degree in mechatronic engineering from the Politecnico di Torino, Turin, Italy, in 2023, where he is currently pursuing the Ph.D. degree with the DAUIN Department.

He is currently a Researcher with DAUIN Department, Politecnico di Torino. His research interests include safety, reliability evaluation, and hardening of AI accelerators