POLITECNICO DI TORINO
Repository ISTITUZIONALE

An Edge AI Industrial Environment Supporting Reliability Assessment and Enhancement

Original
An Edge AI Industrial Environment Supporting Reliability Assessment and Enhancement / Vilar De Farias, G., Celik, A., Rodriguez Condia, J.E., Pau, D., Sonza Reorda, M.. - (2026). (IEEE International Conference on Omni-Layer Intelligent Systems (COINS) Bologna (ITA) 07-09 September 2026) [10.1109/COINS69995.2026.11715563].

Availability:
This version is available at: 11583/3014929 since: 2026-10-22T10:05:49Z

Publisher:
IEEE

Published
DOI:10.1109/COINS69995.2026.11715563

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

18 November 2026

An Edge AI Industrial Environment Supporting Reliability Assessment and Enhancement

Gustavo Vilar de Farias*, Celik Abdullah*, Josie E. Rodriguez Condia*, Danilo Pau†, Matteo Sonza Reorda*
*Politecnico di Torino - Department of Control and Computer Engineering (DAUIN)
†STMicroelectronics

***Abstract*—[1] The expansion of decentralized intelligence has intensified the need for Edge Artificial Intelligence (Edge AI) to provide real-time decision-making, local data privacy, and reduced communication overhead directly on embedded devices. However, the adoption of these solutions relies on development platforms that assist designers in balancing hardware selection and software implementation, specifically by balancing inference performance and memory footprint. In safety-critical domains like robotics and healthcare, designers must integrate reliability into the early design phases of Edge AI systems. This involves estimating the effectiveness of different solutions, quantifying reliability targets, and determining specific methods to improve system resilience. As Edge AI moves into safety-critical sectors, the need for reliable solutions must be balanced against accelerated development cycles. However, the absence of integrated reliability tools in current Edge AI frameworks often forces designers to skip these vital analyses or implement them manually, requiring significant designer effort. Consequently, there is an urgent requirement for platforms that support early-stage reliability assessment and automated hardening. This paper presents a reliability extension to the industrial ST Edge AI Core suite, augmenting the existing toolchain with reliability assessment capabilities. The extension can *i)* estimate the impact of possible faults affecting the inference phase of a Neural Network (NN) *ii)* harden the NN model by introducing software redundancy on its most critical layer. Experimental results on a set of benchmark NNs show that the extended suite can effectively be used to harden (with a very limited impact on the designer effort) an existing model, trading off the achieved accuracy, cost, and reliability.**

I. Introduction

Nowadays, *Artificial Intelligence* (AI) is highly adapted to Edge Computing systems (Edge AI) thanks to advances in transistor miniaturization, the availability of efficient hardware acceleration architectures, and the exploitation of compacting strategies in AI models (e.g, quantization, pruning, and sparsity) [1]–[4].

Commonly, *Software Development Kits* (SDKs) for Edge devices and platforms use specialized programming stacks to support the integration of AI application development and the performance enhancement in these embedded scenarios [5]. However, SDK integration in safety-critical domains imposes additional development challenges, including resilience and functional safety analysis [6].

[1]This work was partially supported by the TIRAMISU project under the EU Grant#101169378 — HORIZON-MSCA-2023-DN-01.

979-8-3195-0489-0/26/$31.00 ©2026 IEEE

In safety-critical domains such as automotive, autonomous robotics, healthcare, surveillance, and aerospace, Edge AI applications must be able to detect and manage system errors to mitigate their potential consequences. In more detail, Edge AI devices, once deployed in the field, are usually exposed to extreme environmental conditions, like temperature variations or exposure to external radiation or Electromagnetic Interference (EMI) sources, which might accelerate aging or directly cause permanent or transient errors. In many domains, such errors can lead to severe consequences, including vehicle collisions, incorrect medical diagnoses, satellite de-orbiting, and system-collapsing failures. Therefore, Edge AI systems deployed in such contexts must treat reliability across hardware, software, and application layers as a primary design concern.

Unfortunately, current edge-class SDKs mostly focus on the development and integration of Edge AI models under performance, area, and accuracy constraints, [7]. In fact, non-functional properties are commonly neglected, including vulnerability assessment utilities and the automatic protection of an application against soft errors. Notably, reliability assessment is often skipped during design due to its computational cost, limiting designers' visibility into system vulnerabilities.

On the other side, it is well known that the cost for matching reliability/safety constraints increases when shifting this analysis to the latest design steps. Hence, it is of paramount practical importance to anticipate reliability analysis and hardening as much as possible during the design flow.

Inspired by constraints in the software stacks of edge-class SDKs, this work proposes and develops a framework that supports the automatic reliability evaluation and the effective hardening of neural networks (NNs) for Edge AI platforms. The proposed solution works on the model of the considered NN, thus anticipating its reliability analysis and possible hardening. The approach assumes that the hardware cannot be modified in order to support reliability or safety, and only acts on the software running on the selected hardware platform.

In detail, the proposed framework aims to assist application designers in reliability analysis and possible hardening of AI applications for their deployment on Edge devices. In particular, the framework takes advantage of the design phase to include early reliability evaluations of a targeted AI application and identify possible fault vulnerabilities in weights. Then, one or more software-based hardening strategies are applied selectively to reduce application vulnerabilities. It is worth noting that both phases, evaluation and protection, consider

the constraints of the target Edge platform. Our framework is developed and evaluated on an automotive-class *ST Edge AI* platform (*Stellar E*). The experimental results on several representative Edge AI applications indicate that reliability evaluations and selective improvements can be applied while considering platform constraints and preserving AI model accuracy and performance.

The document is organized as follows. Section II briefly describes reliability assessment methods from the literature, defines the fault model, and then details the considered hardening techniques and the ST Edge AI platform. Section III describes the solutions adopted for assessing and enhancing the reliability using the ST Edge AI platform. Section IV reports and analyzes the results, and Section V concludes the work and outlines directions for future work.

II. Background

A. Reliability assessment

Characterizing the impact of faults is a fundamental requirement for the reliability analysis of safety-critical systems [8]. The objective of this characterization is to determine vulnerabilities at the application, assessing how faults propagate to affect final inference accuracy. Identifying these vulnerable structures is essential for guiding the deployment of hardening mechanisms. Thus, designers can implement effective fault-tolerance strategies that mitigate system-level risks while limiting overhead, such as memory usage and performance.

In the literature, reliability estimation typically involves comparing a system's response under fault conditions with its fault-free execution. This evaluation can be conducted through physical testing, such as beam experiments that expose the hardware to radiation to induce real-world errors, or through extensive fault injection (FI) simulations [9]–[11]. While beam experiments provide high-fidelity data on hardware resilience, simulation-based FI campaigns are frequently preferred during the design phase due to their cost-effectiveness.

Fault injection simulations can be performed at different levels, ranging from hardware gate-level to software system-level [9], [11], each with its own characteristics and limitations in terms of performance and fidelity to real fault effects. These abstractions of a system's physical defects are called fault models. In general, models close to the physical implementation (e.g., a transistor with an open gate) are more realistic but require high computational power. Conversely, models close to the application (e.g., erroneous multiplications) are faster to simulate, though they may not accurately reflect the fault effect in real systems [9], [10].

More realistic fault models are often prohibitively expensive in AI domains due to the high amount of operations in these workloads [12]. To address this, a methodology that proposes changes to the NN weights [13] provides a balance between simulation performance and physical realism [14], enabling the modeling of some hardware failures like memory bit-flips.

Among the techniques to improve FI performance, a methodology for statistically sampling faults [15] significantly reduces the number of faults simulated (Fault Space) while maintaining a high confidence level in the results.

This work uses bit-flips in NN weights as the fault model, combined with statistical fault sampling, to estimate reliability for AI workloads in a commercial embedded framework.

B. Hardening Techniques for AI

Hardening techniques for deep learning span a broad range of approaches [16], which can be organized into design-based and redundancy-based strategies. While design-based methods exploit existing architectural properties to improve AI robustness, redundancy-based approaches introduce explicit replication to mask faults at the cost of increased overhead. In embedded contexts, where memory and latency budgets are tight, the key challenge is limiting the overall overhead.

At the software and application levels, hardening techniques can be applied directly to the network model without hardware modifications. Typical design-based and redundancy-based strategies include *Selective Triple Modular Redundancy* (**sTMR**) [17] and **Adaptive Clipper** [18], respectively.

On the one hand, **sTMR** replicates in space only the identified critical layers of an AI application (i.e., those highly susceptible layers to fault-induced accuracy loss), which are identified through a FI campaign [19]. In detail, when a fault affects one of the triplicated copies, the majority voter masks the erroneous output, since the two healthy copies outvote it [20]. By limiting replication to critical layers, **sTMR** preserves the fault-masking benefit of a full TMR while keeping memory and latency overheads localized [17], [21]. On the other hand, **Adaptive Clipper** addresses abnormally large activation values at layers with unbounded outputs, such as ReLU, which are caused by fault impacts in network weights. Such large activation values may propagate and amplify through subsequent layers, substantially degrading classification accuracy [18]. In particular, **Adaptive clipper** extracts the absolute maximum activation value of each ReLU layer under fault-free conditions using a representative dataset. Then, it enforces this as a strict upper bound during inference. Thus, fault-induced spikes are suppressed before they can propagate, enhancing reliability with virtually no memory overhead and negligible inference cost [18], [22].

C. ST Edge AI

The ST Edge AI Core[2] is a software technology suite designed to automate the deployment of pre-trained machine learning workloads directly onto embedded devices, such as micro-controllers (MCU). It supports the embedded developers to bridge the gap between complex AI algorithms and resource-constrained MCUs. Such technology enables us to easily import pre-trained models saved with representations like Keras, TensorFlow Lite, and ONNX. The core of the tool then automatically translates these complex NNs into highly optimized, hardware-specific C code.

A defining capability is its support for the Neural-ART Accelerator, a proprietary Neural Processing Unit (NPU). It's

[2] st.com/en/development-tools/stedgeai-core.html

integrated into chips like the STM32N6, capable of 600 GOPS, at 3 TOPS/W for low-power operation. The tools also support MCUs like STM32 and Stellar automotive families, integrating various ARM Cortex M and R CPUs.

The software supports the deployment of model quantized, down to 8-bit or even 1-bit formats. This strict memory optimization ensures advanced AI can run locally on MCUs with minimal embedded RAM and flash memory. Such processing capability, being supported natively at the edge, eliminates reliance on cloud servers, resulting in ultra-low, real-time application latency. In addition, since raw sensor data never leaves the physical device, this local execution inherently guarantees strict data privacy and security.

Furthermore, the framework handles a diverse range of data inputs, including time-series analytics, audio recognition, object detection, instance segmentation, and anomaly detection. Developers benefit from the technology being integrated into the ST Edge AI Developer Cloud, allowing them to remotely benchmark models on real chips. Moreover, the extensive Model Zoo and Services ecosystem offers a comprehensive collection of pre-trained, application-ready algorithms that streamline and accelerate development. Deployment is lossless and introduces no accuracy penalties. Quantitative evaluation is supported through an automatically generated built-in application, which reports multiple metrics using either test data or randomly generated inputs.

In this work, we implement and integrate our proposed reliability assessment strategy into the ST edge AI environment.

III. Adopted Solutions

This section describes the adopted solutions for measuring and improving the reliability of AI models in an Edge AI environment. The proposed solutions aim to improve AI reliability by hardening critical layers identified through a fault campaign. Furthermore, this section is divided into two parts: the first describes the general architecture of the proposed solutions, while the second provides the specific implementation details of the techniques used.

A. General Architecture

The general architecture provides a generic solution to cover different implementations of fault models, reliability metrics, and hardening techniques. Relying on the ST tool (*stedgeai*) to transform a high-level NN model into the deployed model. The method first performs a reliability analysis through a FI campaign to identify the critical layers. These layers are then hardened based on the faulty reports and the user-defined constraints. Finally, the effectiveness of the hardened AI workload is validated through a complementary FI campaign.

The approach consists of 3 main steps detailed below and depicted in Figure 1 (key contributions highlighted in blue).

1) Reliability Analysis: This step performs the reliability analysis of a targeted NN for a chosen fault model. In detail, reliability is evaluated through FI campaigns at the model level with faults simulated in a host device. For each fault, a set of NN inferences is performed using the validation dataset. Finally, the impact of each fault on the accuracy is reported.

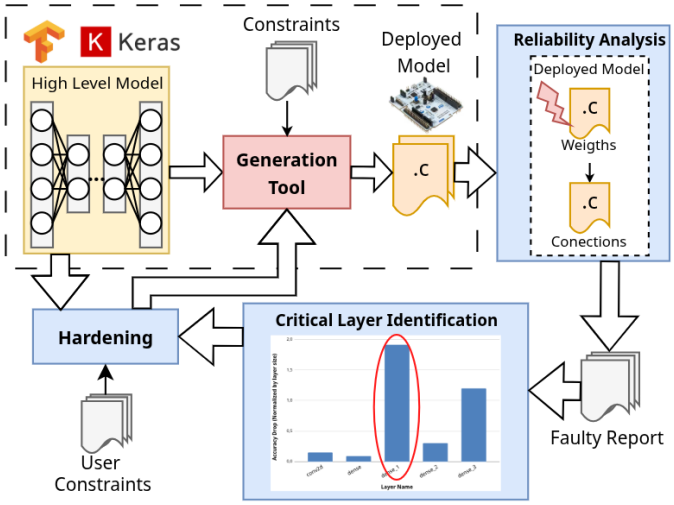


Fig. 1. Scheme of the general architecture

2) Critical Layers Identification: This step identifies the layers where faults have the maximum impact. Hence, hardening these layer(s) maximizes the NN's reliability. Based on the FI results, the target reliability metrics are categorized by layer to assess how faults within a target layer propagate and affect the final inference output.

For each layer, reliability metrics (average and standard deviation) are then calculated and normalized by layer size. This normalized value represents the impact on inference caused by faults arising in that layer. Consequently, the layers that have the greatest impact on the NN results in a faulty scenario are labeled as critical.

3) Hardening: In this step, the high-level NN is modified applying one of the supported hardening techniques described in Section II-B to the critical layer(s) identified in the previous step. The user can select both the layer(s) to be hardened, and the hardening technique to be adopted, thus matching the specific constraints of the application. Then, this hardened version passes again through the generation tool and the reliability analysis step to measure its effectiveness by means of the reliability metrics. Additionally, this step can be repeated until the target metric is achieved.

The user-defined and platform-aware constraints, such as memory or performance limits, as well as reliability thresholds, guide the choice of the hardening technique, since each offers different characteristics in terms of memory, performance, and accuracy overhead. Together with the identified critical-layer information, the hardening process can be selectively applied to satisfy these constraints while reducing overhead.

B. Implementation details

For the implementation, a set of Python programs [3] were integrated into the suite for performing simulation-based fault injection (FI) campaigns in the host, and support sTMR and adaptive clipper (Clip) as hardening techniques. The Stellar E [23] was used as target board for the code generation, featuring

[3] github.com/gvilardefarias/stm-edgeai-reliability

2 MB on-chip flash memory and 2×32-bit Arm Cortex-M7. A total of 8 scripts were developed, totaling 1116 lines of code. The implementation details of each step are detailed below.

1) Reliability Analysis: For this step, permanent stuck-at fault in the NN weight is used as the fault model in the FI campaigns. The script allows the faults to be injected into the deployed model weights at any bit position, weight, and range specified by the user. In detail, the FI script implemented in Python leverages multi-threading by instrumenting the *stedgeai* tool to generate, compile, and validate a faulty NN model. First, a fault-free (Golden) version is generated using the *stedgeai generate* command, producing the deployed NN's C files with the network weights and layer connections. For each fault, a faulty version of the golden weight file is created by applying a bit flip at the target fault location. The faulty models are then compiled and validated using the *stedgeai validate* command, which compares the outputs from the deployed faulty model with those from the high-level golden model and generates an accuracy report.

2) Critical Layers Identification: From the previous step reports, the accuracy drop caused by each fault is extracted and stored in a file organized by fault type and location. Using this information, a graph and a report are generated showing the mean and standard deviation of the accuracy drop (Acc Drop) per layer. This report highlights the impact of faults in the target layer on the final inference accuracy.

Additionally, the Acc Drop is normalized to accurately reflect each layer's contribution to the overall system reliability, as a layer may exhibit high sensitivity to individual faults but have a low global impact due to its size. Moreover, the Acc Drop is multiplied by the ratio between the layer size (e.g., weights per layer) and the total network size (e.g., total weight in the network) ($layer_size/total_size$). Mathematically, this normalized metric represents the layer's proportional contribution to the model's total mean Acc Drop. As such, the sum of these normalized values across all layers equals the model's total mean Acc Drop. Consequently, the layer yielding the maximum normalized Acc Drop is identified as the most critical component for targeted hardening.

3) Hardening: Two hardening techniques are currently supported, and the use can decide which of the two should be adopted on the most critical layer: *i) sTMR*, the target layer is triplicated into three identical clones, and a majority voter layer is used to compare their outputs, *ii) adaptive Clip*, the layer is replaced with a bounded ReLU. To implement the two techniques, we developed a Python program integrated in the suite that modifies the high-level Keras file of the original NN.

In particular, the sTMR generation step creates a new Keras file where the target layer is replaced by a triplicated version and some code implementing a voter is added right after these layers. However, an additional step is required since the *stedgeai* compiler deduplicates identical weight buffers (tmr1, tmr2, and tmr3) during C code generation. Hence, when finding the three truly identical layer copies, the suite collapses them into a single buffer, defeating the triplication. To bypass this compiling feature, a negligible epsilon ($i \times 10^{-6}$, where $i \in \{0, 1, 2\}$) is temporarily added to the weights of each copy, making them numerically distinct and forcing *stedgeai* to generate three separate weight buffers in the C model. A post-processing script then patches the generated weight file by overwriting the tmr2 and tmr3 buffers with the tmr1 values, restoring the three copies to identical weights while preserving the separate buffer allocation required for the redundancy to function correctly at runtime.

For the adaptive Clip, the hardening proceeds in two phases. In the first phase, the training dataset is passed through the original model, and the maximum activation value of each layer is recorded. In the second phase, the model is rebuilt with every ReLU replaced by a bounded *ReLU(max_value* = $v_{max} \times 1.1$*)*, where v_{max} is the recorded maximum and the 10% margin ensures the pre-trained weights remain within the new activation bounds.

IV. Experimental results

To validate the approach, we evaluated 3 representative NNs for Edge computing [4]. The network description (in Keras), their data, and the validation datasets were extracted from the ST modelzoo repository. More in detail, the NNs are: *Human Activity Recognition* (HAR), *Hand Posture* (HP), and *Audio Event Detection* (AED). The HAR is a healthcare application model optimized for edge deployment that classifies accelerometer data from a body-worn sensor into four classes. Similarly, HP is a compact and energy-optimized model that uses time-of-flight sensor data to classify hand posture into 8 classes. In addition, AED is a compact ResNet-based architecture [24] model that classifies input audio into 10 diverse classes. For each NN, Table I presents its architecture and dataset details, including the number of convolution (Conv) and dense layers, as well as the sizes of the total dataset and the validation set.

As a preliminary step, the *stedgeai* tool uses the H5 file to generate compiled C code for each model targeting the Stellar E board. Each model is then verified on the validation dataset.

Using the deployed files, a total of 3 FI campaigns were performed to assess the baseline reliability of the 3 NNs and identify critical layers. All the experiments were conducted on an Intel(R) Xeon(R) Gold 6238R CPU with 252 GB of RAM. The HAR model FI took 5 minutes to complete, the HP 60 minutes, whereas the AED NN lasted about 2.5 days due to the higher architecture and dataset complexity.

More in detail, stuck-at faults were injected into the 16 most significant bits of the NNs weights, with a confidence level of 99% and error margin of 5%. The method described in [15] was used to calculate the confidence level and error margin.

Then, two updated Keras files are generated for each NN by hardening the most critical layer. In our exploration, the first updated version implements sTMR in the target layer, while the second updated model utilizes the Clip technique. Specifically, a single convolutional layer was hardened for the HAR and AED models, whereas a dense layer was targeted

[4] github.com/STMicroelectronics/stm32ai-modelzoo/

TABLE I
ARCHITECTURE AND DATASET DETAILS FOR THE CONSIDERED NNS

NN	Architecture		Dataset	
	Conv	Dense	Validation	Total
HAR	2	1	317	1,585
HP	1	2	404	2,020
AED	6	1	2,289	9,156

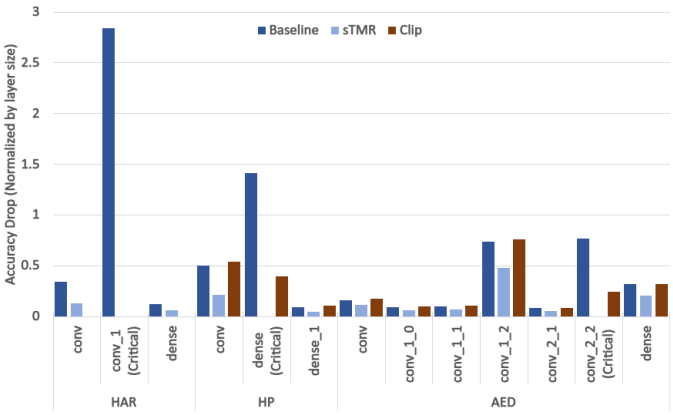


Fig. 2. Normalized accuracy drop per layer for the original, sTMR, and Clip versions of the HAR, HP, and AED models for the Stellar E board.

for the HP network. It is worth noting that the Clip approach cannot be evaluated in the HAR NN because it lacks layers with ReLU activations.

A complementary set of FI campaigns (3 for sTMR and 2 for the Clip models) was performed to validate the effectiveness of the selective hardening approaches. In this case, each FI campaign used the same fault-sampling strategy.

The results of the FI campaign per layer across the 3 models are summarized in Figure 2, which illustrates the normalized accuracy drop per layer with the identified critical layer highlighted. In addition, the plot includes results for the baseline (Base) architecture and hardened versions using both sTMR and Clip techniques.

Table II summarizes the features and the reliability impact of each NN across its versions. The NN features are provided by the *stedgeai analyze* command (for the Stellar E MCU). It includes the memory size (Mem) in bytes, the number of multiply–accumulate operations (Mac) representing the performance (Perf), and the NN's accuracy (Acc). The FI campaign provides the mean Acc Drop and its standard deviation (STD) for the NN inference in the presence of faults on the validation dataset. Together with the Acc Drop, the STD provides an estimate of the maximum expected Acc Drop due to faults.

The baseline results in Figure 2 are used to identify the layer with the greatest normalized Acc Drop (i.e., the most critical layer) and, consequently, generate the hardened version. Experimental data indicate that the most critical layer is architecture dependent. While shallow networks like HAR exhibit concentrated sensitivity in a single layer, more complex models, such as AED, show a more distributed profile. This suggests that as NN depth increases, the proportional contribution of each layer to the total mean Acc Drop decreases.

In Figure 2, results for the sTMR versions confirm that the

TABLE II
FEATURES AND RELIABILITY IMPACT RESULTS FOR THE STELLAR E BOARD ACROSS THE ANALYZED NN VERSIONS

NN		Features			Reliability Impact	
Name	Version	Mem (Bytes)	Perf (MaC)	Acc (%)	Mean Acc Drop (%)	STD Acc Drop (%)
HAR	Base	5,840	67,168	84.44	3.30	13.47
	sTMR	16,208	190,080		0.19	3.07
HP	Base	11,008	8,520	99.13	2.01	12.28
	sTMR	29,696	13,256		0.24	4.14
	Clip	11,008	8,552		1.01	7.30
AED	Base	501,544	7,561,760	84.90	2.23	12.59
	sTMR	797,480	11,697,824		0.97	7.93
	Clip	501,544	7,634,464		1.77	10.18

hardened layer exhibits zero Acc Drop due to fault masking. Furthermore, the increase in the overall model size caused by triplication, as shown in Table II, leads to a decrease in the Acc Drop for the remaining unhardened layers. Since the normalized metric is scaled by the ratio of the layer size to the total network size, expanding the denominator reduces the normalization ratio of unhardened layers in the final report, even though their individual error rates remain unchanged.

Regarding the Clip versions in Figure 2, the hardened layer of each NN shows a decrease in Acc Drop, while the other layers exhibit results close to the Base model. These slight variations in the unhardened layers stem from the transition from ReLU to a bounded activation function, thereby altering the model's error-propagation dynamics. On one hand, the clipping mechanism can improve resilience by suppressing fault-induced activation "spikes" originating in other layers before they can propagate through the network. Conversely, a fault in an unhardened layer might produce a distorted activation that retains sufficient characteristic features for correct inference but is clipped, transforming the fault into a misclassification. However, the results demonstrated that these variations are negligible.

Unlike the *Reliability Impact* section of the Table II, the *Features* values do not require any simulation time. This enables immediate confirmation that the hardened version meets the Mem, Perf, and Acc requirements before proceeding to the reliability analysis. For instance sTMR introduces some overhead in terms of Mem and Perf, while Clip slightly affects the Perf and Acc values. However, the Clip version does not introduce an accuracy difference in the analyzed NNs. In contrast, experimental results show that full TMR imposes 0% of Acc Drop in faulty scenarios with the cost of at least $3\times$ memory and performance.

Moreover, the FI results confirm an improvement in the NN's reliability for both hardening techniques. Across all analyzed NNs, the results highlight the trade-off between feature overhead and fault resilience. It can be seen that the sTMR technique imposes higher Mem and Perf overheads than the Clip version, but results in lower Acc Drop and STD. On the other hand, the Clip version does not incur significant Mem or Perf degradation at the cost of lower fault mitigation.

Typically, the sTMR technique introduces higher overhead

on lightweight networks, as the triplication of even a single critical layer can represent a significant fraction of the total system parameters. This is shown in the HAR model, where sTMR results in a $2.8\times$ increase in both memory footprint and computational performance, whereas in the larger AED network, this overhead is reduced to approximately $1.5\times$.

Interestingly, the experimental results suggest that sTMR hardening in dense layers rather than convolutional ones can result in a performance degradation lower than the memory overhead. This occurs because convolutional layers involve highly complex operations, while dense layers are often more memory-intensive. For instance, while the HAR and AED models exhibit consistent scaling across both network parameters, the HP model (dense layer hardened) shows a clear discrepancy: the memory overhead increases by approximately $2.7\times$ w.r.t. the Base version, while the performance overhead is limited to $1.5\times$. Consequently, the computational overhead of sTMR may be tolerable in architectures where the identified critical layer presents lower operational complexity.

In practice, the optimal choice of hardening technique is driven by the available board memory, along with application-specific constraints such as the maximum inference latency and the threshold for an acceptable accuracy drop. For example, in environments where the HP model requires a minimum accuracy of 94%, the sTMR version is a clear candidate. On the other side, the Clip version achieves only 90.82% accuracy under fault conditions, given its maximum Accuracy Drop of 8.31% (mean + standard deviation). Yet, for more resource-constrained applications, Clip can increase the reliability by a factor of 2 with negligible performance overhead ($1.004\times$).

While sTMR provides superior fault masking, its hardware overhead may exceed the strict resource constraints of embedded environments. Specifically, in memory-intensive architectures like the AED model, triplicating layers can reduce the available flash memory for application logic and increase inference latency beyond real-time requirements. For example, the sTMR version deployed in the STM32C5 MCU, which features only 1 MB on-chip flash memory, will utilize almost 80% of the available memory. In such scenarios, the Clip solution serves as a viable alternative as it enhances system reliability with no memory cost and a negligible impact on performance. Yet, sTMR can be required in more critical applications where accuracy values are stricter.

Overall, the proposed solutions facilitate reliability evaluation and enhancement in systems where such analysis was previously hindered by the lack of integrated evaluation utilities in standard SDKs and the prohibitive computational time typically required for reliability characterization. By providing an automated framework that identifies critical layers during the design phase, this work enables the deployment of hardened Edge AI models in systems where it was previously unfeasible due to memory and performance limitations. Both selective hardening techniques demonstrate significant improvements in application-level resilience against hardware-induced faults while ensuring that memory and performance overheads remain within the strict resource constraints of the target embedded platform. This approach is particularly attractive for resource-constrained applications in safety-critical environments where maintaining high accuracy under fault conditions is a primary concern.

V. Conclusion and future works

This work introduced an extension to an industrial suit for Edge AI applications evaluation and deployment, able to take into account reliability with a limited impact on the designer's efforts. The proposed solution faces reliability evaluation and enhancement for Edge AI applications by leveraging the *stedgeai tool*. The method uses FI campaigns to identify critical layers for software hardening, thereby enhancing system reliability without imposing excessive performance or memory overhead.

The experimental results on 3 representative Edge AI applications demonstrate that NN reliability can be effectively enhanced using two distinct approaches, each presenting a different trade-off between resilience and resource consumption. The designer can select the most suitable based on the specific requirements of the target application. The sTMR technique offers substantial reliability gains by masking errors in critical layers, at the expense of increased memory usage and computational overhead. Conversely, the Clip technique serves as a low-cost hardening solution, providing meaningful protection ($2\times$) with negligible impact on the embedded device's overall memory ($0\times$) and performance ($1.004\times$).

In future works, we plan to extend the work to a wider range of NNs and boards. Furthermore, we will explore fault models and hardware-aware FI methods using the physical board (e.g., via injection of faults into core registers). Furthermore, we will evaluate other hardening techniques, such as DMR with average and quantization with TMR.

References

[1] B. Dally, "Hardware for deep learning," in *IEEE Hot Chips 35 Symp.*

[2] J. Chen *et al.*, "Information compression in the ai era: Recent advances and future challenges," *IEEE Journal on Selected Areas in Communications*, 2025.

[3] P. V. Dantas *et al.*, "A comprehensive review of model compression techniques in machine learning," *Applied Intelligence*, 2024.

[4] U. Kulkarni *et al.*, "Ai model compression for edge devices using optimization techniques," in *Modern Approaches in Machine Learning and Cognitive Science: A Walkthrough: Latest Trends in AI*, 2021.

[5] V. Shankar, "Edge ai: A comprehensive survey of technologies, applications, and challenges," in *2024 1st International Conference on Advanced Computing and Emerging Technologies (ACET)*, 2024, pp. 1–6.

[6] J. Becker, "A safety-certified automotive sdk to enable software-defined vehicles," in *Proc. Workshop Fahrerassistenz und automatisiertes Fahren*, 2022.

[7] T. Sipola, J. Alatalo, T. Kokkonen, and M. Rantonen, "Artificial intelligence in the iot era: A review of edge ai hardware and software," in *2022 31st Conference of Open Innovations Association (FRUCT)*, 2022, pp. 320–331.

[8] A. Maurya *et al.*, "Reliability of safety-critical systems: A state-of-the-art review," *Quality and Reliability Engineering International*, 2020.

[9] F. F. d. Santos *et al.*, "Revealing gpus vulnerabilities by combining register-transfer and software-level fault injection," in *51st Annual IEEE/IFIP International Conf. on Dependable Systems and Networks*.

[10] M.-C. Hsueh *et al.*, "Fault injection techniques and tools," *Computer*, 1997.

[11] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, "Quantifying the accuracy of high-level fault injection techniques for hardware faults," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 375–382.
[12] D. Gnad *et al.*, "Reliability and security of ai hardware," in *2024 IEEE European Test Symp.*
[13] G. Esposito *et al.*, "Comparing application-level hardening techniques for neural networks on gpus," *Electronics*, vol. 14, 2025.
[14] F. Pessia, J.-D. Guerrero-Balaguera, R. L. Sierra, J. E. R. Condia, M. Levorato, and M. S. Reorda, "Effective application-level error modeling of permanent faults on ai accelerators," in *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2024, pp. 1–7.
[15] A. Ruospo *et al.*, "Assessing convolutional neural networks reliability through statistical fault injections," in *2023 Design, Automation Test in Europe Conf. Exhibition*.
[16] C. Bolchini *et al.*, "Resilience of deep learning applications: A systematic literature review of analysis and hardening techniques," *Computer Science Review*, 2024.
[17] T. G. Bertoa *et al.*, "Fault tolerant neural network accelerators with selective tmr," *IEEE Design & Test*, 2022.
[18] L.-H. Hoang *et al.*, "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *2020 Design, Automation & Test in Europe Conf. & Exhibition*.
[19] A. Bosio *et al.*, "A reliability analysis of a deep neural network," in *2019 IEEE Latin American Test Symp. (LATS)*.
[20] R. E. Lyons *et al.*, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, 1962.
[21] A. Ruospo, G. Gavarini, I. Bragaglia, M. Traiola, A. Bosio, and E. Sanchez, "Selective hardening of critical neurons in deep neural networks," in *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2022, pp. 136–141.
[22] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," *IEEE Design Test*, pp. 1–1, 2025.
[23] STMicroelectronics, *Stellar electrification MCUs*. [Online]. Available: https://www.st.com/resource/en/datasheet/sr5e1e3.pdf
[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385