AI-Based Detection and Classification of Adversarial and Fault-Induced Threats in Split Computing

AI-Based Detection and Classification of Adversarial and Fault-Induced Threats in Split Computing

G. Esposito[1] (*†), E. Magliano[1] (*), N. Scarano[1] (*), T. Ahmed Eltaras[1] (*), J.D. Guerrero Balaguera[1], L. Mannella[1], J.E. Rodriguez Condia[1], A. Ruospo[1], S. Di Carlo[1], M. Levorato[2], A. Savino[1], M. Sonza Reorda[1]

[1] *Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy*
name.surname@polito.it

[2] *University of California - Computer Science Department, Irvine, US*
levorato@uci.edu

***Abstract*—Split Computing (SC) has emerged as an effective strategy for deploying Deep Neural Networks (DNNs) on edge and Internet of Things (IoT) platforms. By partitioning inference between constrained edge devices and powerful cloud servers. However, the intermediate feature representations exchanged between the Head and Tail models can be corrupted either by *deliberate* adversarial perturbations, obtained by crafting the input to the Head, or by *unintended* hardware faults affecting the computations in the Head. Both phenomena manifest as distortions in the feature maps, but they have distinct causes and require different responses. Therefore, detecting and classifying such anomalies is crucial for dependable and secure SC-based inference. This work formulates feature-map monitoring in SC as a *three-class classification* problem. A lightweight classifier is trained to distinguish among *clean*, *fault-induced*, and *attack-induced* feature maps at the split boundary. It thereby determines whether corruption is present and identifies its most likely source, enabling the detection and classification of threats. We conduct a detailed analysis of how hardware faults and adversarially crafted inputs in the Head affect the distribution of entries in the intermediate feature map. Performing a design-space exploration across multiple SC configurations. Experiments on SC ResNet-50 show that the SC configuration, which involves intermediate 3-channel feature maps, achieves an F1-score of 97.84% on the clean/adversarial/faulty classification task, with a 31.07% computational overhead relative to baseline SC inference. These results indicate that three-class feature-map classification is a practical and effective front-end for cause-aware analysis in SC deployments.**

***Index Terms*—Split Computing, Deep Neural Networks, Reliability, Security, Adversarial Attacks, Hardware Faults, Edge Intelligence, Threat Detection.**

This work was supported by PNRR – M4C2 – Investimento 1.3, Partenariato Esteso PE00000013 – "FAIR—Future Artificial Intelligence Research" – Spoke 8 "Pervasive AI", funded by the European Commission under the NextGenerationEU programme and through the National Center for HPC, Big Data and Quantum Computing. Also, this work was supported by Project SERICS through the MUR National Recovery and Resilience Plan, funded by the European Union NextGenerationEU under Grant PE00000014, project COLTRANE-V funded by the Ministero dell'Università e della Ricerca within the PRIN 2022 program (D.D.104 - 02/02/2022), and Project Vitamin-V funded by the European Union: Project 101093062. We also acknowledge ISCRA (HardNet) for the access to the LEONARDO supercomputer (EuroHPC JU), hosted by CINECA, Italy.

(*) The authors equally contributed to this work.
(†) Correspondence to: giuseppe.esposito@polito.it.

I. Introduction

Artificial Intelligence (AI) is increasingly integrated in modern SAfety-CRitical real-time Systems (SACRS), where systems must make real-time decisions based on large volumes of sensory data [1]. In this context, Deep Neural Networks (DNNs) have emerged as the dominant computational tool, achieving state-of-the-art performance in perception, prediction, and control. However, edge computing platforms in SACRS and other cyber-physical environments are often constrained by limited processing power, memory, and energy. As a result, running large DNNs directly on these devices is often impractical, motivating distributed inference approaches that spread the computational workload across multiple devices.

Within this context, Split Computing (SC) has emerged as an effective strategy for deploying DNNs across heterogeneous environments [2]. In a typical SC architecture, the network is partitioned into a *Head*, which runs on the resource-constrained device, and a *Tail*, which executes on an edge or cloud server with greater computational capabilities. The Head processes the initial layers of the model and produces intermediate Feature Maps (FMs), which are then forwarded to the Tail for completion of the inference. This collaborative execution enables computationally intensive tasks—such as object detection, navigation, or depth estimation—to be performed on lightweight platforms while still leveraging external compute resources. Introducing a split in the inference pipeline, however, raises significant concerns about system dependability and security. From a *reliability* perspective, the computation performed by the Head can be affected by hardware faults arising from manufacturing variability, environmental conditions, or aging effects [3], [4]. Such faults can corrupt the Head's internal activations and, in turn, the generated FMs, potentially leading the Tail to misclassify without explicit error reporting [5]. From a *security* perspective, adversarial attacks constitute an important threat: an attacker can deliberately craft input to the Head so that the resulting FMs drive the Tail towards incorrect predictions, even though the hardware is functioning correctly. This vulnerability has been widely documented in the adversarial machine learning literature, where carefully designed perturbations to the input have been shown to induce incorrect predictions in deep neural networks while remaining difficult to detect [6], [7]. In the context of

split computing, such attacks are particularly relevant when the input source to the Head, e.g., a sensor or external data stream, can be influenced or manipulated by an adversary. In this work, we treat these two mechanisms as the primary sources of distortion in the intermediate representations. Hardware faults in the Head corrupt the FMs through erroneous computations, whereas adversarial attacks corrupt them by manipulating the Head's input. Although both phenomena ultimately manifest as anomalies in the intermediate FMs, their underlying causes have different operational implications [8]. As a result, it is not enough to flag a FM as "abnormal": the system must also infer whether the anomaly is more likely due to a hardware fault or an adversarially crafted input. A fault-induced distortion indicates a problem with the Head's computational pipeline. This requires reliability-oriented actions, such as performing self-tests or logging the event for potential hardware maintenance or reconfiguration. An adversarial distortion signals that the data source or conditions are unreliable. This requires security measures, such as rejecting the input, down-weighting it, or adopting a more cautious decision policy. Treating all anomalies as a single undifferentiated class either leads to conservative reactions with unnecessary performance penalties or to the application of inappropriate countermeasures.

Beyond the specific case of SC, the broader tension between performance, reliability, and security has been extensively discussed in the literature. The study presented in [8] analyze this triad in advanced CMOS technologies and show that addressing these three aspects jointly remains an open challenge. In particular, they observe that existing design solutions rarely achieve a "sweet spot" where reliability concerns are mitigated, security threats are countered, and performance requirements are met simultaneously. Performance-oriented and reliability-oriented designs typically aim to reduce variability and degradation [9], [10], whereas many security primitives deliberately exploit variability to obtain entropy or to detect misuse [11]. As a consequence, mechanisms that strengthen security can inadvertently weaken reliability, and vice versa, making it difficult to optimise both simultaneously.

Redundancy-based schemes can be employed to detect and, in some cases, correct hardware faults by replicating computations and comparing the DNN output. However, they come at the cost of additional hardware, energy, and area.In resource-constrained environments, the cost of hardening solutions is crucial, and options that require additional hardware or significant edge computation may not be feasible.

Existing work has begun to address the reliability and security aspects of SC systems, but largely in isolation. The reliability of SC architectures has been studied through analytical modeling and extensive fault-injection campaigns [5], [12]–[16], while most security-oriented research in distributed AI has focused on federated learning and collaborative training [17]–[19], rather than run-time inference in split models.

This work extends our preliminary study [20], where we formulated the problem as a binary classification task over the intermediate FMs in SC. In that setting, every analyzed FM was assumed to be corrupted, and the goal was to distinguish whether the distortion was *fault-induced* or *attack-induced*. The study demonstrated that intentional and unintentional corruption leave distinct signatures in the feature space, enabling their separation with a lightweight server-side classifier. However, the binary formulation did not account for the case where the model operates correctly, and the FM is clean, and therefore could not decide whether a corruption is occurring in the first place.

To fill this gap, we reformulate the problem of monitoring SC inference at the split boundary as a **three-class classification** task over the intermediate FMs. The objective is to distinguish, at run time, whether a given FM is (i) *clean*, (ii) corrupted by a hardware fault in the Head, or (iii) produced by an adversarially crafted input. A lightweight classifier is trained on FMs collected under these three conditions and deployed on the server side, where it operates on the same FMs used by the Tail for inference. This approach does not require modifying the underlying DNN architecture and is compatible with supervised-compression schemes integrated into the SC pipeline. Specifically, the main contributions of the paper are as follows:

- We formulate FM monitoring in an SC environment as a detection+classification problem, enabling simultaneous detection of abnormal behavior and attribution of its source of corruption (clean, fault-induced, or adversarial).
- We conduct a detailed analysis of how error models, that mimic the effect of hardware faults, and adversarially crafted inputs, perturb the distribution of FMs extracted from the SC DNN Head, highlighting the distinct corruption patterns each source induces.
- We explore multiple SC configurations based on supervised compression to evaluate how the choice of intermediate FM compression ratio influences both classifier effectiveness and its computational overhead.
- Our findings show that the SC configuration, which compresses the transmitted FM to 3 channels, achieves the best trade-off between the model's F1-score in the clean/adversarial/faulty classification (97.84%) task and the classifier computational overhead (31.07%).

The remainder of this paper is organized as follows. Section II presents background and related work on reliability and security in split inference systems. Section III describes the three-class classification framework, the fault and threat models, and the data collection pipeline. Section IV presents the experimental setup and evaluation protocol. Section V discusses the results and insights. Section VI reports a discussion in which we reflect on how our detection mechanism jointly addresses both reliability-induced faults and security-oriented attacks in SC environments, comparing with previous works. Finally, Section VII summarizes the findings and outlines directions for future work. The developed code to run the experiments will be made available on GitHub (HERE).

II. Background

This section introduces the foundational concepts needed to frame the proposed work. We begin by presenting the principles of supervised compression and SC. We then review the notions of dependability and security according to the classical taxonomy in [21], highlighting how fault-to-error

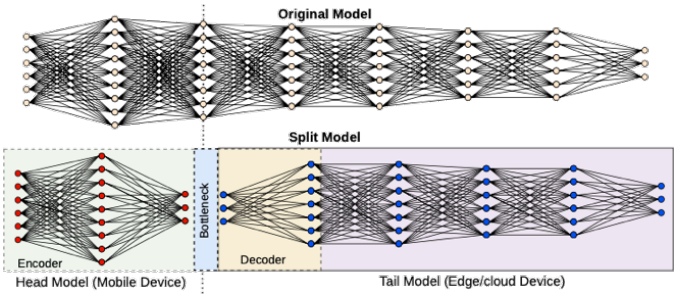


Fig. 1. Supervised Compression for Split Computing [2].

propagation underlies reliability threats. In contrast, exposure to an attack chain defines security threats. Next, we examine hardware fault mechanisms in modern AI frameworks and discuss their impact on the reliability of DNN-based inference. Finally, we describe the main characteristics of adversarial attacks and highlight how the SC broadens the attack surface in distributed AI systems.

A. Supervised Compression in Split Computing

SC distributes the execution of the DNN inference across a resource-constrained device and an external device (e.g., cloud server) with better computational capabilities. As depicted in Figure 1, if the DNN has L layers, it is divided into two parts: the DNN *Head* with l layers (where $l \ll L$) runs on the edge device and generates intermediate FMs. These FMs are then sent to the cloud server, where the DNN *Tail*, consisting of the remaining $L-l$ layers, completes the inference process using the received FMs. Finally, the inference result is returned to the edge device to complete the task [2], [22].

Despite optimizing the deployed workload on the resource-constrained device through the SC paradigm, DNNs produce large intermediate FMs, often exceeding the available wireless bandwidth. *Supervised compression* techniques address bandwidth limitations by utilizing an encoder-decoder architecture, where the encoder is integrated into the DNN Head. At the same time, the decoder is incorporated into the Tail model running on the edge server. One of the most prominent techniques for the supervised compression is the in-network neural compression, i.e., Channel Reduction + Bit Quantization (CRBQ) [23]. The CRBQ encoder-decoder architecture consists of a sequence of convolutional layers, complemented by Rectified Linear Unit (ReLU) activation functions and Batch-Normalization layers. In the encoder, these layers reduce the number of channels, compressing the FMs to a desired number. In the decoder, the same sequence of layer types as the encoder receives the compressed FM from the cloud and decodes it, enabling the Tail model to complete the inference step.

B. Introduction to dependability and security

Dependability is classically defined as the ability of a system to deliver a service that can be justifiably trusted, and to minimise the frequency and severity of service failures [24]. Dependability is usually characterised by attributes, such as availability, reliability, safety, integrity, and maintainability. Security adds further requirements, most notably confidentiality, which focuses on preventing unauthorised access to data and services. While reliability is primarily concerned with the continuity and correctness of service in the event of accidental faults, security deals with protection against intentional actions that may compromise confidentiality, integrity, or availability.

The authors of [21] propose a unified view of dependability and security based on the fault–error–failure chain. A *fault* is a potential cause of an *error*; an *error* is a deviation in the internal state of the system that may lead to incorrect service; and a *failure* occurs when the error propagates to the service interface, causing the delivered service to deviate from its specification. Building on this foundation, subsequent work, such as [25], surveys model-based techniques for evaluating system dependability and discusses how these models are being extended to reason about security as well. Taken together, these contributions highlight that dependability and security can be analyzed within a common conceptual framework, but often with different types of initiating faults (accidental versus intentional) and different evaluation goals.

In safety-critical settings, such as autonomous road vehicles, DNNs face both dependability and security challenges that may compromise the system in similar ways. From the dependability perspective, numerical sensitivity, hardware faults, and hardware aging can introduce silent data corruptions or performance degradation during inference [26], [27]. For instance, soft errors or bit flips in memory or compute units can propagate through the network and significantly alter predictions [26], [28]. From the security perspective, a wide range of adversarial threats has been studied, including adversarial examples [6], data poisoning attacks [29], and model extraction attacks [30], which exploit weaknesses in the training or inference pipeline.

C. Hardware faults and reliability issues in AI-based systems

In the context of DNN inference and hardware accelerators, such as Graphical Processing Units (GPUs), low-level signals can significantly impact the execution of assembly instructions, leading to incorrect algorithm execution, such as memory mismatches in the deployment of a Tiling Matrix Multiplication (TMM) algorithm. The TMM algorithm divides inputs, feature maps, and weight arrays into smaller submatrices called *blocks*, and these submatrices are in turn divided into smaller submatrices called *tiles*. These blocks and consequently the tiles are distributed among the Computational Units (CUs) of a processor/accelerator to compute fragments of matrix multiplications between tiles. However, these TMM tiles might be assigned to faulty CUs.

Consequently, if more than one tile is processed by a faulty CU, a similar error pattern will likely affect the algorithm's results. However, the hardware's structural organization or the implicit nature of the application's code may mask some of these effects, meaning they can go unnoticed if they do not produce visible corruption in the application's outputs or the system itself. Although software mechanisms can sometimes conceal the effects of faults, the main concern is with faults

that propagate silently, resulting in overlooked errors in the Neural Network (NN) 's outputs, also known as Silent Data Corruptions (SDCs) [15], [31], [32].

To study these effects, researchers employ Fault Injection (FI) campaigns that intentionally inject faults into the system to evaluate its resilience under controlled conditions [33]–[35]. In this work, we focus on *software-level FI* as the primary method to assess the resilience of deep neural networks. Software-level FI directly perturbs network parameters, activations, or intermediate tensors, enabling scalable, reproducible, and fine-grained experimentation. Although FI can also be performed at the hardware level [36]–[38], software-based approaches have become the preferred choice in the DNN domain due to their practicality and controllability [14], [33], [39].

In this work, we use the application-level error model from [40], which captures corruption patterns in the intermediate FMs of the General Matrix Multiplication (GeMM) algorithm. This model is based on FI campaigns at the application level with multiple bit-flips, applying patterns of corruption that reflect how errors propagate through hardware, according to Block Error Rate (BlER) and Neuron Error Rate (NER), which indicate the percentage of corrupted blocks and corrupted neurons, respectively. This choice allows our software-level FI to maintain both scalability and realism by preserving the effects of hardware faults on DNN computations.

In contrast, common software-level fault models from the literature directly flip bits in network weights, neuron activations, or convolution outputs during inference [41]. While these models are portable and easy to use across different DNN environments, they overlook the actual propagation of faults from hardware to software. Consequently, they may not accurately replicate fault effects, limiting the evaluation's accuracy. Therefore, we prefer an error model grounded in lower-level FI results to provide a more accurate representation of hardware-induced errors in software.

1) Hardware Faults Detection and Mitigation: Many works in the literature aim to detect and mitigate hardware faults. The HarDNN [42] approach focuses on feature map-level vulnerability evaluation to provide selective protection against hardware errors while avoiding the high overhead of full network redundancy. It uses Δloss as a continuous metric that converges much faster than traditional binary mismatch checks, enabling precise identification of critical components. By hardening only the most vulnerable feature maps, this method achieves resilience improvements, with only 30% additional computation. Other Research on early detection proposes applying mathematical tensor metrics to output feature maps early in the network hierarchy to identify permanent hardware faults [43]. Metrics, such as Minkowski distance and Signal-to-Noise Ratio (SNR), can predict the final impact of a fault, potentially saving significant time and energy. Experimental results show that Minkowski distance is particularly effective, correctly identifying approximately 95% of critical silent data corruption events. Another study evaluates the resiliency of automotive object detection networks to permanent and transient faults using accelerated neutron beam testing on GPU architectures [44]. While hardware-based mechanisms like Error Correction Code (ECC) and parity effectively mitigate transient errors, they are found to be insufficient for permanent faults under the ISO 26262 standard. Consequently, the authors recommend periodic structural tests to ensure safety goals are met within the required fault-tolerant time intervals. The VANDOR strategy [45] addresses single-event upsets in quantized neural networks by forcing erroneous values toward zero, leveraging the observation that models are more resilient to such changes. This hardware-based protection triplicates the sign bit and uses a lightweight voter to manage fault directionality for duplicated sensitive bits. This approach provides high protection efficiency for embedded systems with significantly lower area overhead than traditional triple-modular redundancy.

D. Adversarial Attacks

While hardware faults arise unintentionally, adversarial attacks represent deliberate manipulations of DNN behavior through carefully crafted perturbations. An adversarial attack modifies the input by adding small, often imperceptible changes that cause the model to produce incorrect outputs [6]. Although these perturbations are typically negligible from a human perspective, they exploit the complex, high-dimensional decision boundaries of DNNs, where tiny variations in input space can induce large shifts in output space.

Classical threat models are typically categorized according to the adversary's knowledge of the model in three main categories: *white-box*, *black-box*, and *gray-box* [46], [47]. In a *white-box* setting, the attacker has full access to the model's architecture and parameters and can therefore compute exact gradients via backpropagation to optimize adversarial perturbations. This is the best scenario for an attacker who has full access to all system details. In a *black-box* setting, the attacker has no internal knowledge and must rely solely on observed inputs and outputs, typically using query-based or transfer-based techniques to construct effective perturbations without direct access to gradients. Between these extremes lies the *gray-box* setting, where the attacker has partial information, such as access to only part of the model.

Adversarial attacks on image classification are categorized into several types based on the adversary's knowledge, goals, and the phase of the machine learning lifecycle. Some of the most significant examples from the exhaustive survey [48] are:

- **Fast Gradient Sign Method (FGSM)**: White-box efficient, fast algorithm that uses the gradient of a cost function to generate perturbations.
- **DeepFool**: White-box iterative algorithm that assumes deep learning models are linear and finds the minimum perturbation needed to project an image across the nearest decision boundary.
- **HopSkipJump Attack**: Black-box attack that estimates gradient direction using binary information at the decision boundary.

An emerging and promising field is collaborative AI, in which networks are split across multiple edges. In this paradigm, the Adversarial attacks also follow different rules.

The vulnerability of collaborative paradigms is further highlighted by Fan et al. [49], who introduced the SLADV attack to demonstrate how untrusted servers can exploit intermediate representations in split learning. Demonstrating the feasibility of creating contradictory examples by training shadow models of input levels. The practical vulnerability of split inference is underscored by Rossolini et al. [50], who demonstrated that an adversary with access only to the edge portion of a model can craft universal adversarial perturbations that successfully propagate to an unknown cloud-side component. Their work reveals that manipulating shallow feature representations is sufficient to induce misclassifications, even without knowledge of the full model or decision boundaries.

In SC, the partition of the network into a Head and a Tail naturally induces such a gray-box structure. The Head is deployed on the device and may be more exposed to reverse engineering or inspection. In contrast, the Tail runs remotely (e.g., on a server or cloud platform) and is more difficult for an attacker to access. This motivates a threat model in which the adversary can exploit knowledge of the Head, but the Tail is effectively a black box.

E. Adversarial Attack Detection and Mitigation

For mitigation and rectification, several reactive techniques aim to render perturbations ineffective during inference. Feature Squeezing [51] reduces the search space available to adversaries by applying bit depth reduction or spatial smoothing to the input. Mitigation is achieved by comparing the model's prediction on the original input to its prediction on the squeezed version; a significant difference indicates an adversarial sample, which is then rejected. Perturbation Rectifying Networks (PRN) [52] were designed specifically to counter Universal Adversarial Perturbations. A PRN acts as a pre-input layer that "rectifies" the perturbed image to restore the classifier's original prediction without requiring modifications to the model's parameters.

In the literature, several approaches have been proposed for detecting Adversarial Attacks, which can be grouped into three main categories. Approaches based on external detection models constitute a first-class solution. These methods introduce an additional model that distinguishes adversarial from clean samples by analyzing internal NN representations. The framework by Lee [53] uses FMs extracted from multiple intermediate layers to compute Mahalanobis distance-based confidence scores, identifying samples that deviate from the training distribution, including both Out Of Distribution (OOD) inputs and adversarial examples. Similarly, SafetyNet [54] appends a classifier to the end of a target model to detect distinctive patterns caused by adversarial perturbations in the final layers of ReLU-based architectures. Detector Subnetworks [55] follow the same principle, augmenting the original classifier with subnetworks specifically trained to estimate the likelihood that an input sample is adversarial.

A second group of methods combines detection with input reforming, aiming to move perturbed samples back toward the manifold of clean data. Among these, the MagNet framework [56] employs a detector that evaluates reconstruction error or probability divergence to identify adversarial inputs, while a reformer based on an autoencoder or random noise slightly modifies suspicious samples to facilitate classification.

A third category consists of reactive techniques that mitigate the effect of adversarial perturbations during inference without relying on external classifiers. Feature Squeezing [51] reduces the adversary's degrees of freedom through bit depth reduction or spatial smoothing, and labels an input as adversarial when the predictions for the original and squeezed versions differ significantly. Finally, PRN [52], designed to counter Universal Adversarial Perturbations, operate as a pre-input layer that rectifies the perturbed image to restore the classifier's original prediction without modifying its parameters.

Eventually, to mitigate and rectify adversarial impacts, several reactive techniques focus on rendering perturbations ineffective. Feature Squeezing reduces the adversarial search space using bit depth reduction or spatial smoothing. It compares predictions on the original and modified inputs, rejecting those with significant differences as adversarial samples. Additionally, Perceptual Robust Networks are designed to counter Universal Adversarial Perturbations by rectifying the perturbed image before classification, without changing the model's parameters.

III. METHODS

This section presents the SC environment and the NN used to detect and differentiate corruption sources. Our system determines whether corruption has occurred and, if so, whether it stems from unintentional (i.e., hardware faults) or intentional (i.e., adversarial attacks) perturbations. The methodology relies on the adoption of a neural network model trained for detecting and classifying those perturbations. As shown in Figure 2, the methodology consists of three stages: *i) data collection*, *ii) preprocessing*, and *iii) classifier training and validation*. The pipeline described outlines the offline training process for building a detection model, distinct from the operations executed during runtime in the deployed system.

In the first step, we collect the input decoder FMs during SC DNN inference. At the same time, aside from the golden run, FI campaigns and Adversarial attack simulations are conducted to assess the effects of corruption caused by GPU hardware faults and malicious attacks, respectively. In the second step, we normalize and undersample the data to ensure smooth classifier convergence and reduce redundancy in the generated datasets, respectively. Finally, the third step focuses on classifier training and validation through state-of-the-art metrics in the context of DNN for classification.

A. Data collection

The data collection step comprises the generation of three automatically labeled datasets by collecting intermediate FMs during the SC DNN inference under three different scenarios: *i)*the edge device under hardware physical defect that excites a fault, *ii)* an attacker's violation of the edge device that can corrupt the transmitted intermediate output, and *iii)* the golden run. In the three scenarios, the PyTorch forward hook function extracts the decoder input FMs [57].

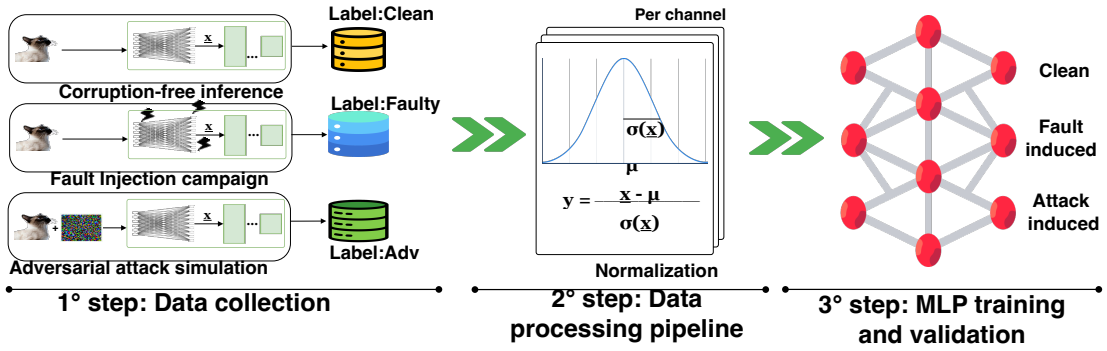


Fig. 2. The main steps of the workflow. Data collection produces three datasets, which are normalised and then used to train the detection and classification MLP.

1) Fault Injection Campaigns: According to the fault model described in subsection II-C, which resorts to the model described in [40], during the FI campaigns, a fault list is generated, specifying the affected layers, Tile Size (TS), BlER, NER, and bit location. The resulting combination of these parameters defines the exact location and the magnitude of the corruption. According to the typical TMM algorithm execution, each FM is divided into blocks; thus, the BlER represents the fraction of corrupted blocks and NER the fraction of corrupted tiles within a block.

In addition, the bit-flip is performed at fixed bit locations, and the overall Bit Error Rate (BER) (i.e., for each convolutional layer, the portion of output FMs that is corrupted during the FI) is computed as in Equation (1).

$$BER = NER \times BIER \tag{1}$$

2) Adversarial Attack Simulation:

a) Threat model: We assume a gray-box adversary in the SC setting. Among the three models described in Section II-D (i.e., white, gray, and black-box models), this model better reflects realistic scenarios in which the on-device portion of the model (i.e., the Head) can be inspected or reverse-engineered by an attacker, while the remote Tail remains protected. This assumption is consistent with prior SC security studies (which used a similar threat model) [50] and with the broader evidence that resource-constrained devices can be compromised by attackers [58], [59], even on a large scale [60].

Accordingly, in this paper, adversarial attacks are modeled as input perturbations applied at the Head. We assume a scenario in which the attacker has full access to the Head model (including its architecture and parameters) but no internal knowledge of the Tail model, which can only be queried as a black box. In particular, the adversary: *(i)* cannot modify the parameters of the Head or the Tail; *(ii)* can inject perturbations only at the input of the Head; *(iii)* can repeatedly query the Tail with different inputs; *(iv)* and the communication channel is assumed to be intact.

The attacker uses knowledge of the Head to guide the perturbation. By iteratively perturbing the Head's input, the attacker aims to alter the intermediate FMs fed to the Tail so that the final prediction becomes a misclassification. In other words, the adversarial behavior is expressed solely through the crafted input. At the same time, the perturbation remains visually negligible at the input level. We focus on an *untargeted* setting, where any change in the original predicted label is considered a successful attack, rather than enforcing a specific target label.

These adversarially induced distortions in the feature maps constitute the second corruption source considered by our three-class attribution framework.

b) Attack procedure: To synthesize adversarial examples under this threat model, we adopt a gradient-guided procedure driven by the norm of the intermediate FMs, summarized in Algorithm 1, following the procedure described in [20]. Given an input sample X, the attacker iteratively refines a perturbed version X^{adv} according to the following steps:

1) **Feature-map computation:** The current adversarial candidate X^{adv} is forwarded through the Head $H(\cdot)$ to obtain the intermediate feature map $F = H(X^{adv})$.
2) **Norm evaluation:** The attacker computes an L-norm (e.g., L_2) of the feature map, denoted $L = \|F\|$, which serves as the objective guiding the perturbation.
3) **Gradient estimation:** The gradient of L with respect to the input X^{adv} is evaluated as in line 5.
4) **Perturbation update:** A small perturbation δ is generated in the direction of the sign of the gradient, scaled by a step size ϵ (line 6), and the adversarial candidate is then updated as in line 7, optionally followed by clipping to keep the input within the valid range.
5) **Tail query:** The updated X^{adv} is passed through the Head, the feature map $H(X^{adv})$ is provided to the Tail $T(\cdot)$, and the corresponding prediction $\hat{y}$ is observed.
6) **Misclassification check:** If the predicted label $\hat{y}$ differs from the original prediction for X, the attack is deemed successful; otherwise, steps 1–5 are repeated until misclassification is achieved, or a maximum number of iterations is reached.
7) **Data collection:** For each successful attack, the feature map at the input of the decoder (i.e., $H(X^{adv})$) is stored and labeled as *adv*. These feature maps constitute the adversarial subset of the dataset used to train the proposed three-class attribution model.

B. Data Preprocessing Pipeline

The second step involves the data pre-processing pipeline, which is separately applied to the data of the three generated

Algorithm 1 Gradient-based adversarial attack in SC

Require: Head model H (white box), Tail model T (black box), input sample X, perturbation step ϵ

$X^{adv} \leftarrow X$ ▷ Initialize with the clean input
repeat
  $F \leftarrow H(X^{adv})$ ▷ Compute intermediate feature map
  $L \leftarrow \|F\|$ ▷ Evaluate norm-based objective
  $\nabla_X L \leftarrow \frac{\partial L}{\partial X^{adv}}$ ▷ Backpropagate through Head
  $\delta \leftarrow \epsilon \cdot \text{sign}(\nabla_X L)$ ▷ Construct perturbation
  $X^{adv} \leftarrow X^{adv} + \delta$ ▷ Update adversarial candidate
  $\hat{y} \leftarrow T(H(X^{adv}))$ ▷ Query Tail prediction
until $\hat{y}$ differs from the original prediction for X or max iterations is reached
return X^{adv}

datasets. To address the variety of dataset generation methods, we ensured that the final training dataset provided to the classifier equally represents each target class, enhancing its generalization capabilities. As described in [61], the undersampling step allows to balance the dataset, preventing the model from becoming biased toward over-represented corruption patterns. We conducted an undersampling process during the classifier training procedure to balance the classes (which is explained in more detail in Section IV). This process was carried out in a stratified manner across all input FMs, ensuring that all the fault and adversarial corruption patterns are equally represented, although the dataset size was reduced. As a result, the original statistical properties of both the clean and corrupted feature-map distributions are preserved, and no artificial distributional shift is introduced at training time.

After balancing the classes, each dataset was partitioned into training (80%), validation (10%), and test (10%) subsets, following a machine learning best practice [61], and a z-score normalization was performed on each dataset partition with the training parameters. During hyperparameter optimization, we used only *60% of the training subset*. We evaluated training portions from 20% to 100% of the training subset and selected 60% as a practical trade-off between optimization cost and final accuracy.

C. Classifier training and validation

One of the main limitations in the literature is the lack of prior work that has specifically studied a detection approach applied in a split-computing environment.

We employed a Multi-Layer Perceptron (MLP) that processes high-dimensional feature maps and assigns the label *Clean*, *Attack-induced*, or *Fault-induced*. The considered architecture consists of two sequential basic blocks, comprising the sequence *i)* Fully connected layer, *ii)* Batch normalization, *iii)* Dropout regularization, and *iv)* ReLU activation function. The two hidden blocks follow the last fully connected layer, which is properly set for the three-class classification. This structure was selected because the input to the MLP is derived from the head model, which maps the data to a latent representation in which the original spatial organization is intentionally discarded. In this setting, convolutional operations would no longer be meaningful, whereas a fully connected architecture can effectively exploit the resulting global feature relations with minimal complexity.

Aligning with [61], in our training setup, we considered the Learning Rate (LR), dropout rate, number of neurons of the MLP hidden layers, Optimized type, LR scheduler, weight decay, momentum (for Stochastic Gradient Descent (SGD)), alpha (for Root Mean Squared proportional (RMSprop)), LR step size, LR scaling factor (i.e., Gamma), and Max Temperature (Max T.). (See table Table I for the selected hyperparameter values).

The training process involved monitoring the training and validation loss and accuracy at each epoch to assess the model's generalization capability. The final model was selected based on the lowest validation loss, ensuring optimal performance on unseen data. After training, the classifier was evaluated on the test set using F1-score and confusion matrices. Each matrix reports, for every class (clean, fault-induced, or attack-induced), the percentage of instances assigned to each possible predicted class. This representation enables a direct assessment of the model's ability to correctly classify samples, while also highlighting specific patterns of misclassifications between classes, providing insight into which types of corruption are more challenging to detect and differentiate.

Once trained, the MLP classifier is intended to be deployed on the SC tail device side (e.g., external server). In this configuration, the classifier receives as input the intermediate FMs produced by the Head and performs corruption detection, while the SC Tail model performs the remaining task. Thanks to this, the tail device can avoid performing the final inference step on the SC tail model using a corrupted FM, which could lead to misclassification and potentially catastrophic results. For this reason, the evaluation of the MLP effectiveness is based on *i)* accuracy-related metrics (True Positive, True Negative, False Positive and False Negative rates, and F1-score), and *ii)* computational overhead introduced by the MLP. This overhead reflects the additional operations added to the Tail model on the edge device for assessing feasibility in the context of SACRSs. In such systems, the number of operations can serve as a proxy for latency overhead [62], and latency remains a primary concern [1].

IV. Experimental Setup

We used the simulation environment from [63] to emulate the SC system locally, including the SC DNN architectures and pre-trained models. Experiments were conducted on six SC configurations of ResNet50 using the ImageNet 2012 dataset [64]. Specifically, FMs were extracted under three conditions: FI, adversarial attacks, and clean inference.

We adopted the CRBQ method, which inserts an encoder–decoder at the split point to compress intermediate FMs via a bottleneck layer. The evaluated configurations were CRBQ(1), (2), (3), (6), (9), and (12) corresponding to the intermediate FM compression to 1, 2, 3, 6, 9, 12 channels, respectively.

A subset of 2,500 images (50 per class across 50 ImageNet instances) was selected to ensure diversity and avoid overfit-

ting. For each configuration, three datasets were generated: faulty, adversarial, and clean.

To conduct the FI campaigns, we utilized a modified version of PyTorchFI [65], which allows the corruption of DNN components during inference according to a user-defined function. This version of PyTorchFI incorporates the error model outlined in Section III-A1[1]. As the first step in the FI campaign, the framework generates a list of experiments to be executed for each campaign. This list includes the parameters TS, $BlER$, NER, and $bit\_location$. The FI campaigns were performed with TS fixed at 16, corresponding to the typical TS managed by a GPU CU. BlER was varied within the interval $[0.2, 1]$ to represent GPU architectures with differing numbers of CUs. NER ranged within $[0.02, 0.1]$, leading to a BER ranging between $[0.4\%, 10\%]$. The bit locations were restricted to the interval $[19, 31]$, as faults in the range $[0, 19]$ have been shown to have a negligible impact [16]. The fault list comprises 1,600 injected faults, and for each fault, three of the 2,500 images are randomly sampled, resulting in a dataset of 4,800 faulty samples for each SC configuration. Similarly, to generate the adversarial dataset, our attack methodology outlined in Paragraph III-A2b has been applied, varying the number of iterations from one to 300. We limited the attack to 300 iterations because, beyond that point, the success rate saturated, while the additional computational cost increased substantially.

The 2,500 input images used for the FI campaign were used for the adversarial attack. Each input image was subject to two different adversarial attacks. These attacks differ in the number of adversarial iterations, as described in Section III-A2. This results in two attack-induced samples for each image, producing a total of 5,000 adversarially perturbed FM.

Aside from these two datasets, a clean dataset composed of 2,500 golden images has also been created.

To balance the three classes, undersampling is performed by reducing the number of faulty and adversarial samples to match the number of golden samples. First, samples containing invalid values (NaNs) are removed to ensure data integrity. Then, from the remaining valid samples, a fixed number is selected via uniform random sampling, resulting in balanced class distributions across the dataset. After undersampling, the complete dataset for each configuration was divided into three sets: 80% (6,000 samples) for training, 10% (750 samples) for validation, and 10% (750 samples) for testing.

Applying random horizontal flips and Gaussian blur (an image-smoothing technique that reduces detail) to the three datasets introduces structured noise, reducing redundancy and enhancing variability. This prevents overly rapid convergence and strengthens generalization.

Hyperparameter optimization was performed using Optuna [66], exploring the search space outlined by the following hyperparameters:

- $h_1 \in \{64, 128, \ldots, 512\}$
- $h_2 \in \{32, 64, \ldots, h_1\}$
- $d \in [0.2, 0.6]$
- $\eta \in [10^{-4}, 5 \times 10^{-3}]$
- $\lambda \in [10^{-6}, 5 \times 10^{-2}]$
- opt $\in \{\text{Adam}, \text{RMSprop}, \text{SGD}\}$

The best configurations are reported in Table I. For all configurations, the number of epochs was set to 15, and the training dataset portion was set to 60%. These values were chosen manually (without Optuna) to balance a good trade-off between model accuracy and training time, rather than solely maximizing accuracy. Increasing the number of epochs or the training portion improved accuracy only marginally. We evaluated training-set portions from 20% to 100% of the training set and found that using 60% provided a practical trade-off between accuracy and training time.

Figure 3 shows validation accuracy across Optuna trials. Later trials generally achieve higher and more stable performance, reflecting effective exploration and convergence.

Experiments were conducted on the HPC cluster at Politecnico di Torino [67], featuring Intel Xeon Gold 6130 CPUs and NVIDIA Tesla V100 GPUs.

V. Results

A. Outlining input samples distribution

Table II summarizes the statistical properties of the FMs fed as input to the MLP across different SC configurations. For each SC configuration, three sample classes are reported: Clean, Fault-induced, and Attack-induced. The table provides, for each class, the mean, standard deviation, maximum, minimum, and Peak signal-to-noise ratio (PSNR) (i.e., one of the reference quality metrics for digitally processed images and video [68]) of the FMs. These statistics provide an overview of the numerical distributions fed to the MLP and enable an initial assessment of class separability.

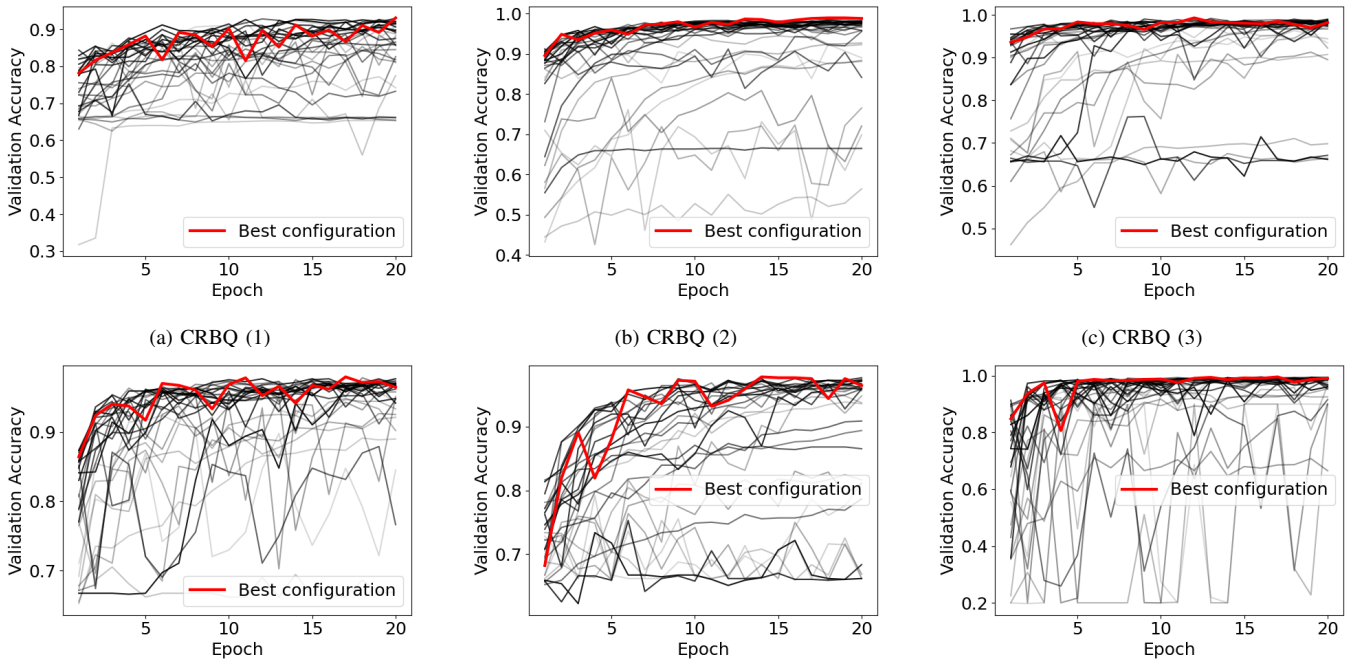
Although PSNR (Peak Signal-to-Noise Ratio) is primarily used to evaluate signal quality, we include it in our analysis not to assess visual fidelity, but to measure how much the intermediate FMs deviate from their expected behavior when not corrupted. To achieve this, aligning with [69], we define a reference FM that represents the system's clean activation profile. This reference is created by averaging all clean feature maps generated under the same Split Computing configuration, aggregated across all input images. We compute the PSNR between each class sample using this clean FM profile as the reference. This statistical prototype captures the typical activation pattern produced by the Head in the absence of corruption, regardless of the input's semantic class. Therefore, PSNR values reported for the clean class quantify how closely clean feature maps align with this reference distribution. In contrast, PSNR values for feature maps affected by faults or adversarial attacks indicate the extent to which they deviate from the expected clean behavior.

The descriptive statistics suggest a high potential for the MLP to distinguish the three classes correctly. The pronounced separation of Fault-induced FMs and the measurable differences between Attack-induced and Clean samples allow the classifier to detect the corruption and identify its source accurately. We therefore hypothesize that the MLP will achieve

[1]To support reproducibility, the modified PyTorchFI scripts implementing the error model used in this work will be released on a public GitHub repository upon acceptance.

TABLE I
HYPERPARAMETER CONFIGURATION FIND BY OPTUNA PER SC CONFIGURATIONS.

Config	*LR*	*Dropout*	*Hidden dim. 1*	*Hidden dim. 2*	*Optimizer*	*LR Scheduler*	*Weight Decay*	*Momentum*	*Alpha*	*Step size*	*Gamma*	*Max T.*
CRBQ(1)	3.44×10^{-4}	0.10	80	80	Adam	Step LR	6.74×10^{-3}	–	–	45	0.43	–
CRBQ(2)	2.08×10^{-3}	0.43	32	64	Adam	Step LR	1.13×10^{-5}	–	–	15	0.46	–
CRBQ(3)	5.23×10^{-2}	0.58	128	48	SGD	Step LR	5.36×10^{-5}	0.17	–	50	0.18	–
CRBQ(6)	9.17×10^{-4}	0.19	64	120	RMSProp	Cosine LR	2.26×10^{-4}	–	0.94	–	–	30
CRBQ(9)	1.91×10^{-3}	0.31	192	96	SGD	Cosine LR	6.60×10^{-4}	0.84	–	–	–	82
CRBQ(12)	1.96×10^{-3}	0.36	80	112	SGD	Step LR	2.95×10^{-3}	0.85	–	50	0.30	–



(a) CRBQ (1) (b) CRBQ (2) (c) CRBQ (3)

(d) CRBQ (6) (e) CRBQ (9) (f) CRBQ (12)

Fig. 3. Training curves for CRBQ configurations across epochs for multiple Optuna trials. Each curve represents a different hyperparameter configuration (lighter lines: earlier trials; darker lines: later trials), while the red line highlights the best configuration.

strong performance due to the inherent statistical differences between the three classes.

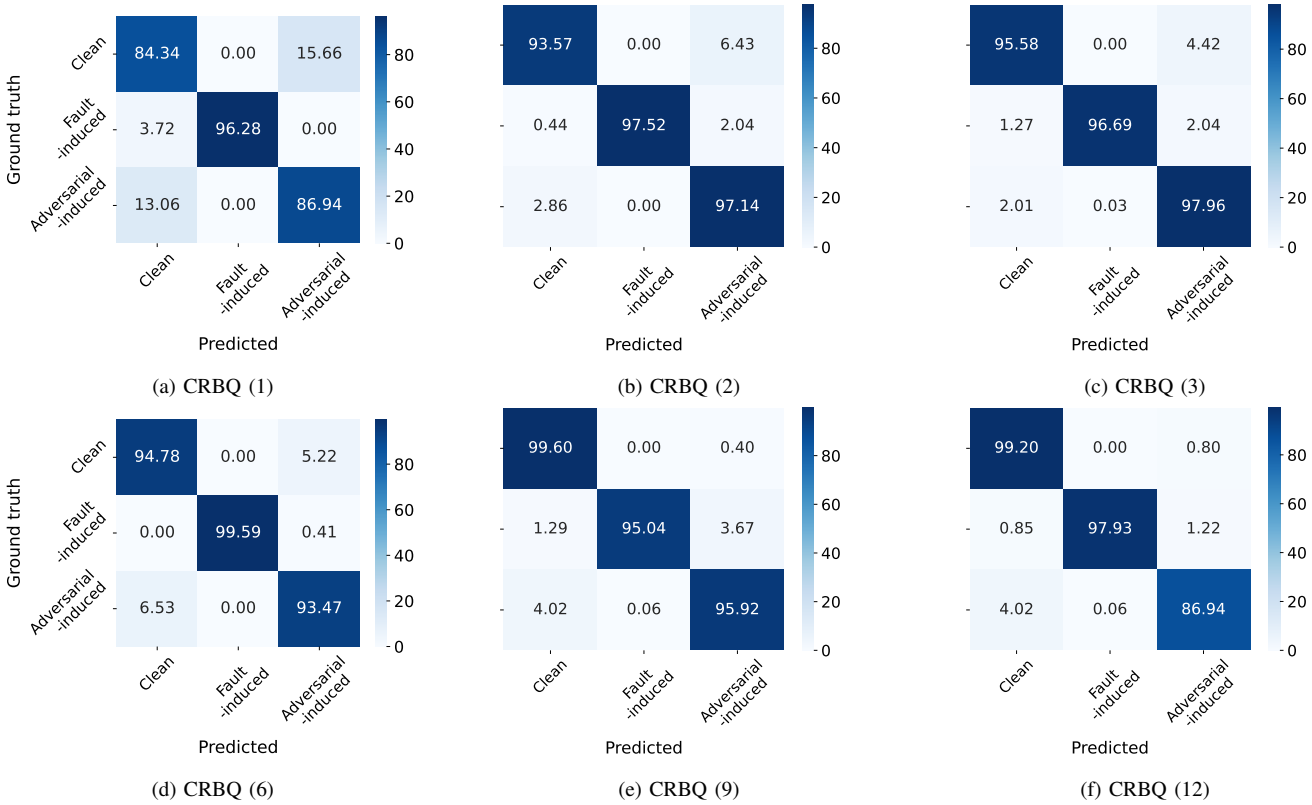
B. MLP performance

To assess the model's ability to detect corrupted FMs and correctly identify their source (clean, fault-induced, attack-induced), we reported in Figure 4 the confusion matrices in the MLP classification task for FMs coming from each SC configuration. This representation enables a direct comparison across configurations for the MLP ability to classify the input samples and highlights class-oriented trends.

Across the six configurations (CRBQ (1), CRBQ (2), CRBQ (3), CRBQ (6), CRBQ (9), CRBQ (12)), the confusion matrices show a progressive increase of True Positive percentage as the number of channels in the FMs increases from 1 to 12. In the first configuration, CRBQ (1), the classifier correctly identifies 84.34% of clean samples, 96.28% of fault-induced samples, and 86.94% of attack-induced samples. Although the model achieves high accuracy for the fault-induced class, clean and attack-induced class samples still exhibit noticeable overlap, as reflected by the 13.06% of clean samples and 15.66% of attack-induced samples assigned to each other's class. This indicates that a single-channel feature representation captures only a limited portion of the features necessary for correct discrimination.

With CRBQ (2), correct classifications increase substantially across all classes: 93.57% for clean, 97.52% for fault-induced, and 97.14% for attack-induced samples. False positive and False Negative percentages decrease significantly (down to 6.43% for clean samples misclassified as attack-induced samples), demonstrating that even increasing the channel dimensionality by one provides FMs that enable more reliable separation among corruption classes. This trend continues in CRBQ (3) and CRBQ (6), where the True Positive rate reaches 95.58% in classifying clean samples.

The highest degree of alignment along the diagonal appears in the later configurations, CRBQ (9) and CRBQ (12). In CRBQ (9), the classifier correctly labels 99.60% of clean samples, 95.04% of fault samples, and 95.92% of attack-



(a) CRBQ (1) (b) CRBQ (2) (c) CRBQ (3)

(d) CRBQ (6) (e) CRBQ (9) (f) CRBQ (12)

Fig. 4. Normalized Confusion Matrices for CRBQ configurations. Rows correspond to ground-truth labels (Attack-induced, Fault-induced, Clean), and columns represent predicted labels. Each confusion matrix is row-normalized: every entry is divided by the total number of samples belonging to the corresponding ground-truth class. Therefore, each row sums to 100% up to rounding. The matrix highlights the classifier's ability to detect and distinguish corruption sources, with diagonal values indicating correct predictions and off-diagonal values revealing misclassification patterns.

induced samples. CRBQ (12) exhibits equally strong performance, achieving 99.20% correct classification for clean samples, 97.93% for fault-induced ones, and 86.94% for attack-induced ones. The small residual fraction of attack-induced samples misclassified as clean can be attributed to the nature of the gradient-based adversarial attacks used in our setup. These attacks explicitly optimize perturbations to produce FMs that are as indistinguishable as possible from benign ones, thereby minimizing detectable deviations in the intermediate representations. As a consequence, a limited subset of attack-induced examples succeeds in shaping their feature activations to closely mimic clean patterns, making them inherently more difficult for the classifier to distinguish.

VI. Discussion: A Joint System-Level Analysis for Split Computing Deployment Environments

Previous research [8] has demonstrated that mechanisms designed to enhance security can unintentionally compromise reliability, and vice versa. This dual challenge makes it difficult to optimize both aspects simultaneously. Therefore, it is essential to classify the source of any detected corruption in order to implement targeted solutions that address both reliability and security issues.

Our work addresses the problem of reliability and security by providing a corruption detection mechanism that differentiates between reliability faults and security issues. This distinction is crucial for implementing the right recovery methods [8] in SACRS, where inference time is critical. The SC paradigm adds latency to DNN inference due to wireless transmission, first for sending intermediate FMs from the Head, and then possibly for retransmitting the final output. Thus, timely and accurate detection of an external module running alongside the Tail is essential for proper SACRS operation.

From a deployment perspective, the proposed design integrates a corruption detector with minimal impact on the SC pipeline's inference latency. The normalization step and MLP process the received intermediate feature maps concurrently with the Tail model's execution. This means the added latency is limited to the small computational overhead from the MLP and the lightweight normalization layers. Therefore, this solution can operate on an external edge server alongside the Tail without significantly affecting inference time. It allows the system to meet the stringent timing requirements of safety-critical SC inference pipelines while effectively detecting corrupted feature maps before they reach the decision stage. This balance between detection capability and low computational overhead makes the approach viable for real-world applications.

Currently, there is no unified approach for detecting and classifying both hardware faults and adversarial attacks during inference in SC systems. Existing solutions tackle reliability and security separately. For instance, [43] focuses on identifying permanent hardware faults in DNNs through statistical analysis of intermediate feature maps, achieving 94% fault coverage but only addressing reliability. On the other hand,

TABLE II
FEATURE STATISTICS PER CLASS ACROSS DIFFERENT SC CONFIGURATIONS. EACH SC CONFIGURATION INCLUDES: ATTACK-INDUCED, FAULT-INDUCED, AND CORRUPTION-FREE (CLEAN) SAMPLES.

SC Config	*Class*	*Mean*	*Std*	*Max*	*Min*	*PSNR*
CRBQ(1)	*Clean*	-0.02	0.66	6.95	-3.57	4.62
	Fault-induced	0.01	0.23	8.09	-6.66	**53.09**
	Attack-induced	-0.02	0.67	**7.98**	**-3.27**	3.96
CRBQ(2)	*Clean*	0.03	0.72	6.43	-6.37	3.33
	Fault-induced	0.02	0.65	3.13	-10.71	**4.90**
	Attack-induced	0.02	0.77	**4.97**	**-7.28**	2.51
CRBQ(3)	*Clean*	-1.09×10^{-03}	0.70	6.96	-4.91	3.63
	Fault-induced	1.77×10^{-03}	0.11	3.77	-2.84	**74.17**
	Attack-induced	4.84×10^{-03}	0.74	**7.94**	**-7.44**	2.88
CRBQ(6)	*Clean*	0.02	0.66	5.60	-5.69	4.16
	Fault-induced	-1.54×10^{-03}	0.12	6.78	-7.37	**65.55**
	Attack-induced	0.02	0.72	**8.98**	**-9.11**	3.33
CRBQ(9)	*Clean*	4.58×10^{-03}	0.71	6.00	-5.80	3.42
	Fault-induced	-1.11×10^{-04}	0.15	8.52	-11.02	**71.20**
	Attack-induced	3.92×10^{-03}	0.74	**20.17**	**-19.70**	2.95
CRBQ(12)	*Clean*	-9.36×10^{-03}	0.69	6.47	-7.06	3.67
	Fault-induced	-1.67×10^{-03}	0.19	10.21	-10.43	**66.20**
	Attack-induced	-0.01	0.72	**9.68**	**-7.87**	3.20

TABLE III
OPERATIONS (OPS) OVERHEAD AND F1-SCORE ACHIEVED BY THE PROPOSED CORRUPTION-DETECTION MODULE ACROSS THE SIX SPLIT COMPUTING CONFIGURATIONS. THE TABLE HIGHLIGHTS THE TRADE-OFF BETWEEN COMPUTATIONAL COST ON THE EDGE DEVICE AND THE RESULTING DETECTION AND CLASSIFICATION PERFORMANCE. THE PERFORMANCE OF THE PROPOSED APPROACH IS COMPARED WITH TWO REPRESENTATIVE APPROACHES PROPOSED IN PREVIOUS STUDIES FOR FAULT DETECTION [70] AND ADVERSARIAL ATTACKS DETECTION [53].

SC configuration	Detection + classification (This work)		Faults detection [70]		Adversarial attacks detection [53]	
	OPs overhead	**F1-score**	**OPs overhead**	**F1-score**	**OPs overhead**	**F1-score**
CRBQ 1	26.59%	90.04%	0.02%	85.10%	178.1%	74.34%
CRBQ 2	26.74%	96.66%	0.04%	44.57%	178.2%	64.47%
CRBQ 3	31.07%	97.84%	0.06%	66.67%	178.3%	84.58%
CRBQ 6	33.33%	96.09%	0.12%	66.67%	178.8%	87.80%
CRBQ 9	34.32%	98.51%	0.18%	66.67%	180.0%	89.67%
CRBQ 12	34.52%	98.65%	0.24%	87.46%	184.6%	86.57%

[71] presents ML-Filter, a framework for detecting gradient-based data poisoning attacks during training. It uses Laplacian Deviation Metrics and Clustering to accurately identify poisoned datasets (99.63% for known, 98% for unknown attacks) but is limited to the training phase and does not consider hardware faults or security during runtime inference.

Table III summarizes the trade-offs between detection performance and computational overhead for the considered approaches across the evaluated SC configurations. The rows of the table report the different SC configurations, which correspond to increasing numbers of channels in the compressed intermediate FMs (from CRBQ 1 to CRBQ 12). The columns compare three alternative solutions: the proposed detection and classification framework, the first phase of the reliability-oriented detection approach adapted from [70], and the adversarial attack detection method adapted from [53].

For fairness, all compared methods were evaluated on the same SC configurations, using the same balanced datasets, train/validation/test partitions, and F1-score computation protocol. The reliability-oriented baseline [70] was adapted to operate on the decoder-input feature maps for each SC configuration, while the adversarial-detection baseline [53] was adapted to the same intermediate representations and evaluated under the same corruption scenarios considered for the proposed method.

For each approach and configuration, the table reports both the operational overhead in terms of additional operations (OPs) introduced with respect to the baseline inference pipeline and the resulting F1-score achieved in the corruption detection task. Overall, the results show that the three approaches exhibit significantly different trade-offs between computational cost and detection capability, reflecting the distinct design objectives of the respective methods.

The proposed detection and classification framework consistently performs well across all SC configurations with moderate computational overhead. In the most constrained setting (CRBQ 1), it achieves an F1-score of 90.04% with an overhead of 26.59%. As the number of channels increases, performance rises, peaking at 98.65% in CRBQ 12. This improvement is due to richer feature representations in higher-dimensional intermediate feature maps, though it also leads to a slight increase in computational overhead from 26.59% to 34.52%. Overall, the method balances detection accuracy and computational efficiency.

The reliability-oriented detection method from [70] demonstrates a different trade-off, with overhead consistently below 0.25%. This low overhead stems from simple statistical metrics, but it limits the method's effectiveness. The F1-scores vary, peaking at 87.46% in CRBQ 12 but remaining substantially lower than the proposed approach in most configurations.

In contrast, the adversarial attack detection method from [53] shows high F1-scores, reaching 89.67% in CRBQ 9. However, this comes with a significant computational overhead, exceeding 178% of the baseline cost, and reaching 184.6% in CRBQ 12. The high overhead is due to the additional processing required for adversarial detection, making this approach less suitable for latency-sensitive SC systems where fast inference is crucial.

The comparison highlights that the three approaches prioritize different design objectives and therefore exhibit distinct trade-offs between computational overhead and detection capability. The reliability-focused method minimizes computational cost but provides limited detection performance in the considered multi-corruption scenario, while the adversarial detection approach achieves higher accuracy at the expense of a substantial computational overhead. In contrast, the proposed framework offers a balanced compromise between these two extremes, delivering consistently high detection and classification performance while keeping the additional computational

cost within a moderate range. This balance makes the proposed approach particularly suitable for practical deployment in SC-based SACRSs, where both runtime efficiency and reliable corruption detection are essential requirements.

VII. Conclusion

In this work, we presented a novel method for detecting feature maps corrupted by a hardware fault or an adversarial attack on a resource-constrained device. By analyzing the behavior of the method across six SC configurations (CRBQ) with increasing channel dimensionality, we highlighted that richer FMs (i.e., with a higher number of channels) improve the discriminability of corrupted patterns at the cost of increasing classifier complexity.

Our results show that CRBQ(3) achieves the best trade-off between computational overhead and detection performance, introducing 31.07% additional operations while reaching an F1-score of 97.84%. This configuration offers a practical balance for SACRS, maintaining high detection accuracy while keeping computational costs manageable. Overall, the proposed method demonstrates that integrating a corruption-detection and classification mechanism into environments where SC DNN are deployed is feasible and effective, ensuring the security and reliability of SACRSs.

References

[1] R. Pellizzoni, P. Meredith, M.-Y. Nam, M. Sun, M. Caccamo, and L. Sha, "Handling mixed-criticality in soc-based real-time embedded systems," in *Proceedings of the seventh ACM international conference on Embedded software*, 2009, pp. 235–244.

[2] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–30, 2022.

[3] S. Hamdioui, D. Gizopoulos, G. Guido, M. Nicolaidis, A. Grasset, and P. Bonnot, "Reliability challenges of real-time systems in forthcoming technology nodes," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 129–134.

[4] M. Badaroglu, "More moore," in *2021 IEEE International Roadmap for Devices and Systems Outbriefs*, 2021, pp. 01–38.

[5] G. Esposito, J.-D. Guerrero-Balaguera, J. E. R. Condia, M. Levorato, and M. Sonza Reorda, "Assessing the reliability of different split computing neural network applications," in *2024 IEEE 25th Latin American Test Symposium (LATS)*. IEEE, 2024, pp. 1–6.

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[7] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[8] F. Rahman, D. Forte, and M. M. Tehranipoor, "Reliability vs. security: Challenges and opportunities for developing reliable and secure integrated circuits," in *2016 IEEE International Reliability Physics Symposium (IRPS)*, 2016, pp. 4C–6–1–4C–6–10.

[9] K. J. Kuhn, M. D. Giles, D. Becher, P. Kolar, A. Kornfeld, R. Kotlyar, S. T. Ma, A. Maheshwari, and S. Mudanai, "Process technology variation," *IEEE Transactions on Electron Devices*, vol. 58, no. 8, pp. 2197–2208, 2011.

[10] W. Wang, S. Yang, S. Bhardwaj, R. Vattikonda, S. Vrudhula, F. Liu, and Y. Cao, "The impact of nbti on the performance of combinational and sequential circuits," in *Proceedings of the 44th annual Design Automation Conference*, 2007, pp. 364–369.

[11] A. Maiti, V. Gunreddy, and P. Schaumont, "A systematic method to evaluate and compare the performance of physical unclonable functions," in *Embedded systems design with FPGAs*. Springer, 2012, pp. 245–267.

[12] J.-D. Guerrero-Balaguera, I. A. Harshbarger, J. E. R. Condia, M. Levorato, and M. Sonza Reorda, "Reliability estimation of split dnn models for distributed computing in iot systems," in *IEEE 32nd International Symposium on Industrial Electronics (ISIE)*, 2023, pp. 1–4.

[13] E.-W. Caro-Anzola, G. Esposito, J.-D. Guerrero-Balaguera, A.-F. Jiménez-López, F.-R. Jiménez-López, R. Limas-Sierra, G. Ramirez-Espinosa, E. Giusto, B. Montrucchio, O.-F. Vera-Cely *et al.*, "IoT and Edge Computing: Applications and Reliability Implications," in *IEEE 26th Latin American Test Symposium (LATS)*, 2025, pp. 1–10.

[14] A. Ruospo, G. Gavarini, C. De Sio, J. Guerrero, L. Sterpone, M. Sonza Reorda, E. Sánchez, R. Mariani, J. Aribido, and J. Athavale, "Assessing convolutional neural networks reliability through statistical fault injections," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[15] H. D. Dixit *et al.*, "Silent data corruptions at scale," *arXiv preprint arXiv:2102.11245*, 2021.

[16] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 2017, pp. 1–12.

[17] Y. Wu, C. F. Chiasserini, F. Malandrino, and M. Levorato, "Enhancing privacy in federated learning via early exit," in *Proceedings of the 5th workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*, 2023, pp. 1–5.

[18] W. Wan, Y. Ning, S. Hu, L. Xue, M. Li, L. Y. Zhang, and H. Jin, "Misa: Unveiling the vulnerabilities in split federated learning," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2024, pp. 6435–6439.

[19] T. A. Eltaras, Q. Malluhi, A. Savino, S. D. Carlo, and A. Qayyum, "R-conv: An analytical approach for efficient data reconstruction via convolutional gradients," in *Web Information Systems Engineering – WISE 2024*, M. Barhamgi, H. Wang, and X. Wang, Eds. Singapore: Springer Nature Singapore, 2025, pp. 271–285.

[20] G. Esposito, E. Magliano, N. Scarano, T. Ahmed Eltaras, J. G. Balaguera, L. Mannella, J. Rodriguez Condia, A. Ruospo, S. Di Carlo, M. Levorato, A. Savino, and M. Sonza Reorda, "AI-Based Classification of Adversarial Attacks vs. Hardware Fault Corruptions in the Split Computing Context," in *2025 IEEE 31st International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2025, pp. 1–7.

[21] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.

[22] A. E. Eshratifar, A. Esmaili, and M. Pedram, "Bottlenet: A deep learning architecture for intelligent mobile cloud computing services," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2019, pp. 1–6.

[23] Y. Matsubara and M. Levorato, "Neural compression and filtering for edge-assisted real-time object detection in challenged networks," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 2272–2279.

[24] K. S. Trivedi, D. S. Kim, A. Roy, and D. Medhi, "Dependability and security models," in *2009 7th International Workshop on Design of Reliable Communication Networks*. IEEE, 2009, pp. 11–20.

[25] D. M. Nicol, W. H. Sanders, and K. S. Trivedi, "Model-based evaluation: from dependability to security," *IEEE Transactions on dependable and secure computing*, vol. 1, no. 1, pp. 48–65, 2004.

[26] Y. e. a. Ibrahim, "Soft errors in dnn accelerators: A comprehensive review," *Microelectronics Reliability*, 2020.

[27] P. Omland, M. Paulitsch, G. Hinz, and A. Knoll, "Towards an understanding of deep neural network resiliency to hardware faults," in *European Dependable Computing Conference*, 2025.

[28] S. e. a. Qutub, "Hardware faults that matter: Understanding and estimating the safety impact of hardware faults on object detection dnns," in *Dependable Computing Conference*, 2022.

[29] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *ICML*, 2012.

[30] F. Tramèr, F. Zhang, A. Juels, M. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *USENIX Security*, 2016.

[31] P. H. Hochschild *et al.*, "Cores that don't count," in *Proc. of the Workshop on Hot Topics in Operating Systems*, 2021.

[32] A. Singh, S. Chakravarty, G. Papadimitriou, and D. Gizopoulos, "Silent data errors: Sources, detection, and modeling," in *2023 IEEE 41st VLSI Test Symposium (VTS)*. IEEE, 2023, pp. 1–12.

[33] A. Benso and S. Di Carlo, "The art of fault injection," *Control Engineering and Applied Informatics*, vol. 13, pp. 9–18, 12 2011.

[34] A. Benso, A. Bosio, S. Di Carlo, and R. Mariani, "A functional verification based fault injection environment," in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, 2007, pp. 114–122.

[35] A. Vallero, D. Gizopoulos, and S. Di Carlo, "Sifi: Amd southern islands gpu microarchitectural level fault injector," in *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2017, pp. 138–144.

[36] E. Magliano, A. Carpegna, A. Savino, and S. D. Carlo, "A micro architectural events aware real-time embedded system fault injector," in *2024 IEEE 25th Latin American Test Symposium (LATS)*, 2024, pp. 1–6.

[37] E. Magliano, A. Savino, and S. Di Carlo, "Real-time embedded system fault injector framework for micro-architectural state based reliability assessment," *Journal of Electronic Testing*, pp. 193–208, 2025.

[38] J. Chen, G. Esposito, F. F. dos Santos, J.-D. Guerrero-Balaguera, A. Kritikakou, M. Krstic, R. L. Sierra, J. E. R. Condia, M. S. Reorda, M. Traiola *et al.*, "Reliability Assessment of Large DNN Models: Trading Off Performance and Accuracy," in *2024 IFIP/IEEE 32nd International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2024, pp. 1–10.

[39] A. B. Göğebakan, E. Magliano, A. Carpegna, A. Ruospo, A. Savino, and S. D. Carlo, "Spikingjet: Enhancing fault injection for fully and convolutional spiking neural networks," in *30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2024, pp. 1–7.

[40] J.-D. Guerrero-Balaguera, J. E. R. Condia, M. Levorato, and M. Sonza Reorda, "Evaluating the reliability of supervised compression for split computing," in *42nd VLSI Test Symposium (VTS)*, 2024, pp. 1–6.

[41] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, "A Reliability Analysis of a Deep Neural Network," in *2019 IEEE Latin American Test Symposium (LATS)*. IEEE, 2019, pp. 1–6.

[42] A. Mahmoud, S. K. S. Hari, C. W. Fletcher, S. V. Adve, C. Sakr, N. R. Shanbhag, P. Molchanov, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Hardnn: Feature map vulnerability evaluation in cnns," *ArXiv*, vol. abs/2002.09786, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:211259500

[43] V. Turco, A. Ruospo, E. Sanchez, and M. Sonza Reorda, "Early Detection of Permanent Faults in DNNs Through the Application of Tensor-Related Metrics," in *2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*, 2024, pp. 13–18.

[44] A. Lotfi, S. Hukerikar, K. Balasubramanian, P. Racunas, N. Saxena, R. Bramley, and Y. Huang, "Resiliency of automotive object detection networks on gpu architectures," in *2019 IEEE International Test Conference (ITC)*, 2019, pp. 1–9.

[45] W. Guillemé, A. Kritikakou, Y. Helen, C. Killian, and D. Chillet, "Vandor: Mitigating seus into quantized neural networks," in *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2024, pp. 1–6.

[46] H. Baniecki and P. Biecek, "Adversarial attacks and defenses in explainable artificial intelligence: A survey," *Information Fusion*, p. 102303, 2024.

[47] H. Khazane, M. Ridouani, F. Salahdine, and N. Kaabouch, "A holistic review of machine learning adversarial attacks in IoT networks," *Future Internet*, vol. 16, no. 1, p. 32, 2024.

[48] S. Y. Khamaiseh, D. Bagagem, A. Al-Alaj, M. Mancino, and H. W. Alomari, "Adversarial deep learning: A survey on adversarial attacks and defense mechanisms on image classification," *IEEE Access*, vol. 10, pp. 102 266–102 291, 2022.

[49] M. Fan, C. Chen, C. Wang, W. Zhou, and J. Huang, *On the Robustness of Split Learning Against Adversarial Attacks*, 09 2023.

[50] G. Rossolini, T. Baldi, A. Biondi, and G. Buttazzo, "Edge-only universal adversarial attacks in distributed learning," 2025. [Online]. Available: https://arxiv.org/abs/2411.10500

[51] W. Xu, D. Evans, and Y. Qi, "Feature squeezing mitigates and detects carlini/wagner adversarial examples," 2017. [Online]. Available: https://arxiv.org/abs/1705.10686

[52] N. Akhtar, J. Liu, and A. Mian, " Defense Against Universal Adversarial Perturbations ," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, 6 2018, pp. 3389–3398. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00357

[53] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 7167–7177.

[54] J. Lu, T. Issaranon, and D. Forsyth, "Safetynet: Detecting and rejecting adversarial examples robustly," 2017. [Online]. Available: https://arxiv.org/abs/1704.00103

[55] E. Biju, A. Sriram, P. Kumar, and M. M. Khapra, "Input-specific attention subnetworks for adversarial detection," 2022. [Online]. Available: https://arxiv.org/abs/2203.12298

[56] D. Meng and H. Chen, "Magnet: a two-pronged defense against adversarial examples," 2017. [Online]. Available: https://arxiv.org/abs/1705.09064

[57] PyTorch, "Pytorch forward hook function," https://pytorch.org/docs/stable/generated/torch.nn.modules.module.register_module_forward_hook.html . Last visited on March 14, 2025.

[58] D. Canavese, L. Mannella, L. Regano, and C. Basile, "Security at the Edge for Resource-Limited IoT Devices," *Sensors*, vol. 24, no. 2, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/2/590

[59] D. Papp, Z. Ma, and L. Buttyan, "Embedded systems security: Threats, vulnerabilities, and attack taxonomy," in *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, 7 2015, pp. 145–152.

[60] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1093–1110.

[61] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[62] G. Esposito, J.-D. Guerrero-Balaguera, J. E. R. Condia, and M. Sonza Reorda, "Comparing Application-Level Hardening Techniques for Neural Networks on GPUs," *Electronics*, vol. 14, no. 5, p. 1042, 2025.

[63] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt, "Supervised compression for resource-constrained edge computing systems," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2022, pp. 2685–2695.

[64] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 2009, pp. 248–255.

[65] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari, "PyTorchFI: A Runtime Perturbation Tool for DNNs," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2020, pp. 25–31.

[66] O. Contributors, "Optuna tutorial documentation," https://optuna.readthedocs.io/en/stable/tutorial/index.html.

[67] Politecnico di Torino, "HPC@PoliTO," http://www.hpc.polito.it.

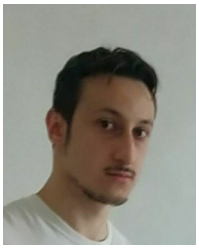
[68] D. Poobathy and R. M. Chezian, "Edge detection operators: Peak signal to noise ratio based comparison," *IJ Image, Graphics and Signal Processing*, vol. 6, pp. 55–61, 2014.

[69] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," *Advances in neural information processing systems*, vol. 31, 2018.

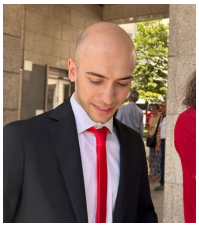
[70] V. Turco, N. Bellarmino, A. Ruospo, R. Cantoro, and E. Sanchez, "DOC: Detection of On-Line Failures in CNNs," in *2025 IEEE 26th Latin American Test Symposium (LATS)*, 2025, pp. 1–6.

[71] L. Sanapala and L. Gondi, "Mitigating gradient-based data poisoning attacks on machine learning models: A statistical detection method," *Indian Journal of Science and Technology*, vol. 17, no. 21, pp. 2218–2231, 2024.

Giuseppe Esposito received the M.Sc. degree in Data Science and Engineering from the Politecnico di Torino, Italy, in 2023, where he is currently a Ph.D. student at the Department of Control and Computer Engineering. His main research topic focuses on designing fault-tolerant AI-based systems in resource-constrained environments. His research interests also include the reliability of parallel architectures and the testing of distributed systems. He is an IEEE student member.



Enrico Magliano is a PhD student in artificial intelligence at the Department of Control and Computer Engineering at the Politecnico di Torino. His research focuses on artificial intelligence for the trustworthiness of computing systems. He also investigates fault-injection methodologies to evaluate and enhance the reliability of embedded applications and explores the robustness of neuromorphic neural networks.



Nicola Scarano received the M.Sc. degree in data science from Politecnico di Torino, Turin, Italy, in 2023, where he is currently pursuing the Ph.D in AI and cybersecurity. His research interests include safe, trustworthy AI for cybersecurity applications.

Tamer Ahmed Eltaras received the M.Sc. degree in electronics from Politecnico di Torino, Turin, Italy, in 2018, where he is currently pursuing the Ph.D. degree in Control and Computer Engineering. His research interests include privacy and security aspects of AI systems, with a focus on distributed and federated learning.

Juan David Guerrero Balaguera received the M.Sc. degree in electronics engineering from Universidad Pedagogica y Tecnologica de Colombia (UPTC), Sogamoso, Colombia, in 2017 and the Ph.D. degree in Computer Engineering from Politecnico di Torino, Turin, Italy, in 2024. He is now a postdoctoral research fellow at the same institution. His research interests include digital design, fault-tolerant AI hardware, functional tests, artificial intelligence, parallel architectures, reconfigurable computing, and FPGAs. He is a member of the IEEE.

Luca Mannella is a post-doctoral fellow at the Department of Control and Computer Engineering (DAUIN) of Politecnico di Torino (Italy) and a member of the IEEE. He holds a B.Sc., M.Sc., and Ph.D. in Computer Engineering from Politecnico di Torino. During his Ph.D., he primarily focused his research activities on supporting developers in creating more secure Internet of Things solutions, with a specific focus on smart home systems and Smart Home Gateways. He is currently also working on the reliability and security of automotive systems.

Josie E. Rodriguez Condia received the M.Sc. degree in electronics engineering from Universidad Pedagogica y Tecnologica de Colombia (UPTC), Sogamoso, Colombia, in 2017 and the Ph.D. degree in Computer Engineering from Politecnico di Torino, Turin, Italy, in 2021. He is now an Assistant professor at the same institution. His research interests include functional testing, architecture of domain-specific accelerators, parallel architectures, Graphics Processing Units, and embedded system design. He is an IEEE member.

Annachiara Ruospo received the M.Sc. degree in computer engineering from the Politecnico di Torino, Italy, in 2018, where she is currently an Assistant Professor with the Department of Control and Computer Engineering. 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. She is a Member of the AI Existential Safety Community of the Future of Life Institute.

Stefano Di Carlo received an M.Sc. degree in computer engineering and a Ph.D. in information technologies from Politecnico di Torino, Italy, where he is a Full Professor. His research interests include dependable and secure computing, neuromorphic computing, and emerging computing paradigms. He has coordinated several national and European research projects. Di Carlo has published over 270 papers in peer-reviewed IEEE and ACM journals and conferences. He regularly serves on the Organizing and Program Committees of major IEEE and ACM conferences. He is a Golden Core member of the IEEE Computer Society and a senior member of the IEEE. He currently serves on the editorial board of JETTA.

Marco Levorato Marco Levorato (Senior Member) is a Professor in the Computer Science department at the University of California, Irvine. He completed the PhD in Telecommunication Engineering at the University of Padova, Italy, in 2009. Between 2010 and 2012, he was a postdoctoral researcher jointly at Stanford and the University of Southern California. Prof. Levorato's research interests are focused on distributed computing over unreliable wireless systems, especially for autonomous vehicles and robotic applications. In this area of research, he has more than 170 papers in IEEE and ACM venues. His research is funded by the National Science Foundation, the Department of Defense, Intel and Cisco. In 2020-2021, he was the vice chair of the IEEE Technical Committee on Smart Grid Communications. He serves in the TPC of IEEE Infocom, IEEE Secon, IEEE Percom, IEEE ICDCS and ACM MobiHoc, is an editor of the IEEE/ACM Transactions on Networking, and was part of the organizing committee of several IEEE and ACM conferences.

Alessandro Savino (M'14, SM'22) is an Associate Professor in the Department of Control and Computer Engineering at Politecnico di Torino (Italy). He holds a Ph.D. (2009) and an M.S. equivalent (2005) in Computer Engineering and Information Technology from the Politecnico di Torino in Italy. His research contributions include Approximate Computing, Hardware Security, Emerging Technologies in AI, Reliability Analysis, Safety-Critical Systems, Software-Based Self-Test, and Image Analysis.

Matteo Sonza Reorda received an M.Sc. degree in electronics and a Ph.D. degree in Computer Engineering from Politecnico di Torino, Italy, in 1986 and 1990, respectively, and is currently a full professor at the Department of Control and Computer Engineering. He published more than 400 papers on test and fault-tolerant design of reliable circuits and systems, receiving several Best Paper Awards at major international conferences. He is involved in research projects with companies and research centers worldwide. He is an IEEE Fellow.