AI-based Classification of Intentional vs. Unintentional Corruptions in the Split Computing context

AI-based Classification of Adversarial Attacks vs. Hardware Fault Corruptions in the Split Computing context

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*
surname@uci.edu

***Abstract*—Split Computing has emerged as a promising paradigm for deploying Deep Neural Networks in Edge and Internet of Things systems, enabling inference tasks to be distributed between resource-constrained edge devices and cloud servers. This approach is particularly attractive for autonomous systems, where security and reliability may be critical. However, intermediate feature maps transmitted between devices are vulnerable to corruption, which may result from intentional adversarial attacks or unintentional hardware faults. Distinguishing whether corruption originates from an external adversary or an inherent system fault is crucial for implementing appropriate countermeasures—reinforcing security mechanisms against attacks or improving system reliability to mitigate the effects of hardware-related faults. To the best of our knowledge, this work is the first to propose a machine learning-based classification mechanism capable of differentiating adversarial attacks from hardware defects in Split Computing systems. The proposed approach analyzes the intermediate feature maps transmitted from the edge device to the server, classifying the source of corruption to guide appropriate responses. Experimental results demonstrate that one of the proposed classifiers can distinguish between intentional and unintentional feature map corruptions with an accuracy of 93.91%.**

***Index Terms*—Internet of Things (IoT), Split Computing (SC), Reliability, Security, Computer Vision, Anomaly classification**

I. Introduction

Artificial Intelligence (AI) is increasingly utilized even in safety-critical systems, often relying on Deep Neural Networks (DNNs) to process large amounts of sensor data in real time for decision making. In particular, within Internet of Things (IoT) environments, Commercial-Off-The-Shelf (COTS) components are commonly employed due to their interesting trade-off between computational resources and cost. To address possible computational constraints, optimizing DNN inference through Edge Computing paradigms is essential to enable efficient deployment.

Among the various optimization techniques in edge computing, Split Computing (SC) has emerged as a promising DNN design paradigm that distributes the computing effort between different devices [1]. This approach partitions the model into two segments: the *Head*, which comprises the initial layers and runs on resource-constrained devices, and the *Tail*, which consists of the remaining layers and operates on more powerful devices, such as cloud or edge servers. By leveraging this architecture, SC enables resource-constrained devices to perform complex deep learning tasks (e.g., run-time obstacle detection and depth estimation).

However, despite its advantages, deploying SC DNNs in safety-critical applications introduces significant challenges concerning reliability and security. From a reliability standpoint, modern embedded and IoT devices are increasingly susceptible to hardware faults. These faults may arise from internal defects, such as premature aging, or external factors, including radiation effects [2], [3]. Since reliability is paramount in safety-critical environments, these vulnerabilities can pose substantial risks, potentially leading to erroneous system behavior or failure.

At the same time, security vulnerabilities in IoT systems have gained growing attention [4]. Specifically, in SC-based autonomous vehicles, adversaries can exploit the communication channel between the Head and Tail, injecting malicious corruptions into the intermediate feature maps. Such attacks can disrupt critical perception tasks, leading to object detection, lane following, and traffic sign recognition misclassifications—ultimately compromising system safety.

Interestingly, reliability and security are deeply interconnected in IoT systems. Prior research has demonstrated that security breaches can significantly impact system reliability, potentially leading to operational failures [5]–[7]. Conversely,

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

979-8-3315-3334-2/25/$31.00 ©2025 IEEE

an unreliable system may introduce vulnerabilities that attackers can exploit. This interdependence underscores the need for a comprehensive approach that jointly considers reliability and security in SC environments.

Despite the growing importance of reliability in distributed systems (e.g., IoT systems) [8], only a limited number of studies have evaluated the fault resilience of SC architectures. The authors of [9], [10] have proposed theoretical and experimental frameworks to assess the reliability of SC DNNs. In general, faults can silently propagate as errors through the DNN execution, jeopardizing the application reliability [11]–[13]. Similarly, research on security threats in distributed deep-learning applications has predominantly focused on federated learning [14]–[16], which is designed for DNN training rather than real-time inference in SC environments. Consequently, a research gap remains in understanding and mitigating security threats, specifically within SC-based real-time inference systems. This gap is particularly concerning in adversarial attacks, where attackers can exploit the Head-Tail communication to manipulate the feature maps and force the DNN into failure. Such targeted attacks could disrupt critical tasks, leading to severe consequences, especially in autonomous and safety-critical applications.

These challenges highlight the urgent need for further investigation into the reliability and security of SC architectures. Both aspects can independently or jointly lead to failures in DNN execution, resulting in catastrophic consequences. While previous studies have extensively examined the detection of corruption in DNN-based systems [17], identifying the source of such corruption—whether from hardware faults or adversarial attacks—remains an ongoing challenge. Moreover, this unexplored challenge is essential for implementing the appropriate mitigation strategies to ensure system safety and reliability. For example, in an autonomous vehicle, after adapting a DNN to the split computing paradigm and deploying the Head on the mobile device, if a hardware defect occurs, the original DNN architecture can be fully deployed on a cloud server. This would help guide the autonomous vehicle to a safe location, although it might increase latency temporarily due to the need to transmit raw data from onboard sensors [1]. On the other hand, if there is an adversarial attack corrupting data transmitted over the wireless connection, the wireless channel can be encrypted [18]. Meanwhile, a lightweight DNN can be deployed entirely on the mobile device at the cost of a temporary decrease in the accuracy of the DNN.

To address these challenges, this paper presents a machine learning-based classification mechanism to distinguish adversarial attacks from hardware defects in SC systems. The key contributions of this work are as follows.

- We propose an adversarial attack that corrupts intermediate feature maps with limited knowledge (only the Head model), demonstrating its feasibility within SC-based systems.
- We analyze the effect of hardware faults, such as bit flips and computation errors, on SC-based inference. We show how these errors propagate through the pipeline and lead to misclassifications in deep learning models.
- We develop a classification model capable of differentiating adversarially corrupted feature maps from those affected by hardware faults, providing a method for identifying the source of corruption.

The remainder of the paper is organized as follows: Section II covers background on the reliability and security threats in the SC context. Section III describes the employed strategy to distinguish intentional and unintentional attacks. The experimental setup is presented in Section IV, followed by the experimental results in Section V. Finally, Section VI outlines conclusions and future remarks of our work.

II. Background

A. Fault to error propagation

Hardware systems, such as Graphical Processing Units (GPUs), can experience faults due to physical manufacturing defects. The effects of these faults can propagate through the various hardware components composing the entire system. A fault can corrupt data, instructions, or control flow when it reaches the software layer. The resulting errors may silently propagate during the execution of a DNN, compromising the reliability of the application [12]. Although different system components (both software and hardware) can mask the effects of faults [19], understanding error propagation is crucial for reliability engineers. This knowledge enables them to develop efficient and effective methods for simulating error behavior, leading to faster and more realistic simulations for reliability assessment than traditional approaches.

Fault Injection (FI) tools are commonly used to evaluate the reliability of a system [20]. This process involves intentionally introducing faults into a system and collecting metrics to assess the resulting system performance. FI can utilize error models that replicate the effects of faults by altering the system's parameters at the software level. For instance, performing single bit-flips in the weight parameters or the neurons of a DNN is a well-known error model that simulates the impact of faults in the hardware running the application [21]–[23].

B. Adversarial attacks

Adversarial attacks exploit the vulnerabilities of Deep Neural Networks (DNNs) by introducing carefully crafted perturbations to input data, leading to incorrect model predictions while remaining imperceptible to humans [24]. These attacks are typically classified into white-box and black-box settings [25], [26]. In a white-box attack, the adversary fully knows the model's architecture and parameters, enabling precise, gradient-based perturbations. In contrast, black-box attacks rely solely on query-based interactions, making them more challenging to execute.

In Split Computing (SC) systems, adversarial attacks introduce a new attack surface by targeting intermediate feature maps rather than raw inputs. Unlike conventional attacks,

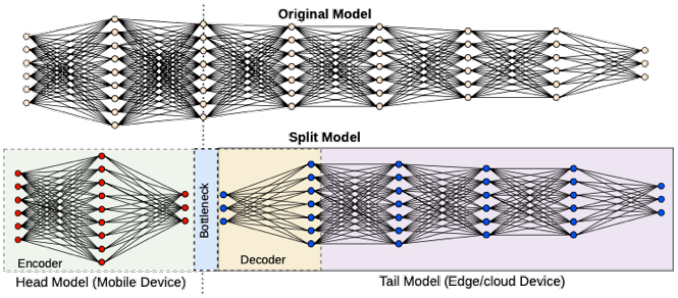


Fig. 1: Supervised Compression for Split Computing [1].

which manipulate the raw input data, an attacker in an SC system can exploit feature representations transmitted from the Head model to the Tail model, leading to misclassification.

C. Supervised Compression for Split Computing

SC distributes the execution of the DNN inference across multiple resource-constrained devices (e.g., mobile device) and external devices (e.g., cloud server) with better computational capabilities. As depicted in Figure 1, assuming that the DNN contains L layers, the original DNN is split into two sub-models: the DNN *Head*, consisting of l layers ($l \ll L$), and the DNN *Tail*, including the remaining $L-l$ layers. The *Head* of the model is deployed on the mobile device, generating intermediate Feature Maps (FMs) that are transmitted via a wireless connection to the cloud server. On the server side, the *Tail* completes the inference process using the received FMs as input. Finally, the inference result may be returned to the mobile device to complete the task [1], [27].

Despite optimizing the deployed workload on the mobile device through the SC paradigm, DNNs produce large intermediate FMs, often exceeding the bandwidth in wireless transmission. *Supervised compression* techniques address the available bandwidth limitations by incorporating in the SC DNN with an encoder-decoder architecture where the encoder is integrated into the DNN Head, while the decoder is incorporated into the Tail model executed by 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) [28]. 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 of channels. In the decoder, the sequence of convolutional layers, along with ReLU activation functions and Batch-Normalization layers, receives the compressed FM from the cloud and decodes it, enabling the Tail model to complete the inference step.

III. Proposed Anomaly Classification Pipeline

This section describes the adopted strategy for distinguishing the types of corruption, whether due to an adversarial attack or a physical hardware defect. Assuming that the collected fault metrics are corrupted, our system aims to determine whether the corruption is due to hardware faults or adversarial attacks. As Figure 2 illustrates, our methodology comprises 3 main steps: *i) Data collection*, *ii) Data preprocessing pipeline*, and *iii) Classifier training and validation*. In the first step, we collect the input decoder FMs during SC DNN inference. At the same time, fault Injection campaigns and Adversarial attack simulations are performed to simulate the effect of corruptions due to GPU permanent faults and malicious attacks, respectively. In the second step, we normalize and undersample the data to guarantee a smooth classifier convergence and to 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 two automatically labeled datasets by collecting intermediate FMs during the SC DNN. The SC system is compromised due to: *i)* the hardware physical defect that excites a fault and *ii)* an attacker violation to the wireless connection that can corrupt the transmitted intermediate output, separately. In both scenarios, the PyTorch forward hook function extracts the decoder input FMs [29].

1) Fault Injection Campaigns: We inject Multiple Bit-Flips according to a given Bit Error Rate (BER) during the Fault Injection Campaigns. We target output FMs of the Head SC DNN, and the injected faults are obtained by extracting a BER fraction of neurons out of the available bit-space. In particular, we use the hardware-aware fault model presented in [22] that simulates faults in the Fused-Multiply Accumulate (FMA) cores of a GPU's Streaming Multiprocessor executing tiling matrix multiplication. Before parameter corruption, a fault list is generated, specifying the affected layers, Tile Size (TS), Block Error Rate (BIER), Neuron Error Rate (NER), and bit location. The resulting combination of these parameters defines the exact location and the magnitude of the corruption. According to the typical Tiling Matrix Multiplication algorithm execution, each FM is divided into blocks, thus the BIER representing the fraction of corrupted blocks and NER the fraction of corrupted neurons within a block as illustrated in Equation (1).

$$BIER = \frac{\#target\_block}{\#total\_blocks}, NER = \frac{\#target\_neurons}{\#total\_neurons} \quad (1)$$

In addition, the bit-flip is performed at fixed bit locations, and the overall BER is computed as Equation (2).

$$BER = NER \times BIER \quad (2)$$

2) Adversarial Attacks Simulation: **Threat Model.** In our setting, we assume a gray-box adversarial model, where the attacker can access the Head model but can only interact with the Tail model through query-based interactions without knowledge of its architecture or parameters. The attacker cannot modify any model parameters in the Head or Tail

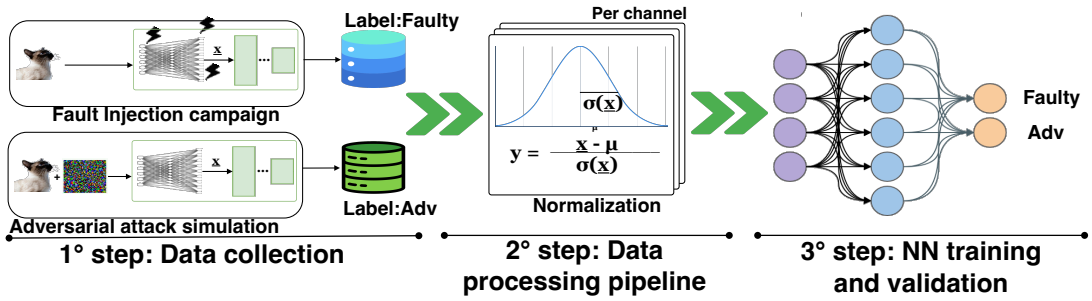


Fig. 2: Main steps of the workflow.

model but can iteratively perturb the input to the Head model, altering the intermediate feature map representation while keeping the modifications imperceptible. This work focuses on an untargeted attack, which aims to cause an incorrect classification rather than enforcing a specific misclassification. The steps of the attack are as follows:

1) **Feature Map Extraction:** The attacker feeds an input sample X into the Head model to obtain the intermediate feature map $H(X)$.
2) **Norm-Based Perturbation Guidance:** The attacker computes the L-norm $||H(X)||$ of the feature map to guide the perturbation process.
3) **Gradient Computation:** The gradient of the input X with respect to the L-norm of the intermediate feature map is computed: $\nabla_X ||H(X)||$
4) **Input Modification:** The attacker slightly modifies the input X in the direction of the computed gradient to maximize the change in the intermediate feature map.
5) **Querying the Tail Model:** The perturbed input X' is passed through the Head model, and the resulting feature map is transmitted to the Tail model for classification.
6) **Misclassification Check:** The attacker checks whether the model's output label has changed. If misclassification is not achieved, steps 2–5 are repeated until the label is misclassified.
7) **Data storage** When the corruption achieves the SC DNN misclassification, the generated input is passed through the Head, and the resulting FM is labeled as *adv* and stored in the corresponding dataset.

The steps described above are formalized in **Algorithm 1**.

Algorithm 1 Gradient-Based Adversarial Attack on Split Computing

Require: Head model H, Tail model (not known) T, Input sample X, Perturbation bound ϵ

Initialize $X^{adv} \leftarrow X$ ▷ Start with original input
repeat
  Compute intermediate feature map $F = H(X^{adv})$
  Compute L-norm: $L = ||F||$
  Compute gradient: $\nabla_X L = \frac{\partial L}{\partial X^{adv}}$
  Generate perturbation: $\delta \leftarrow \epsilon \cdot \text{sign}(\nabla_X L)$
  Apply perturbation: $X^{adv} \leftarrow X^{adv} + \delta$
  Transmit $H(X^{adv})$ to Tail model T and observe predicted label
until misclassification is achieved
Return X^{adv}

B. Data Preprocessing Pipeline

The second step involves the data pre-processing pipeline, which is separately applied to the data of both generated datasets. We conducted an undersampling process to balance the classes and enhance the effectiveness of the dataset (which is explained in more detail in Section IV). Additionally, we implemented a stratified dataset split to avoid overfitting and ensure the generalization of the results. As a result, the datasets are combined in such a way that 80% of the extracted feature maps corresponding to different categories (i.e., one sample per label) are used for training the classifier. The remaining 20% of the feature maps are reserved for validation and testing.

To standardize the data, we performed a per-channel z-score normalization, preserving the statistical information of each channel and fitting them into a standard normal distribution (having zero mean and unitary variance) to improve learning performances. The normalization formula is described in Equation (3).

$$y = \frac{X - \mu}{\sigma(X)} \tag{3}$$

where X is the original channel, μ is the channel mean, $\sigma(X)$ represents the standard deviation of the channel and y is the normalized channel.

C. Classifier training and validation

The classifier used in this study is a Fully Connected Neural Network designed to process high-dimensional feature maps and classify them into two categories (Adv or Faulty). The architecture consists of three sequential basic blocks: a fully connected layer, batch normalization, dropout regularization, and a ReLU activation function.

The classifier is trained using a mini-batch gradient descent approach with a fixed batch size of 32 to guarantee fast training algorithm convergence. The optimization is performed using the Adam optimizer by setting the corresponding hyperparameters to guarantee effective model convergence. Adam is widely used for training fully connected neural networks due to its ability to adapt the learning rate for each

TABLE I: Accuracy of the SC Resnet50 DNNs available in [30] in the corruption-free scenario for each SC configuration corresponding to a specific Channel Reduction (CR).

Accuracy (%)					
CR(1)	***CR(2)***	***CR(3)***	***CR(6)***	***CR(9)***	***CR(12)***
60.79	72.59	73.59	75.5	75.70	75.5

parameter, addressing the varying scales of weights across layers. Adam allowed our classifier for fast convergence by combining Momentum and Root Mean Square Propagation to stabilize updates and accelerate training while guaranteeing convergence. 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 to assess its generalization capabilities. Accuracy, Recall, and Precision metrics evaluate a classifier's performance, especially in handling class imbalances and distinguishing corruption from faults or adversarial attacks. Accuracy measures the proportion of correctly classified instances among all instances. Precision measures the proportion of instances predicted as "adv" that truly belong to this class. In contrast, recall assesses how many of the actual "adv" instances are correctly identified.

IV. Experimental Setup

We used the simulation environment provided in [30] to locally simulate a SC system working environment providing the SC DNN architectures and the pre-trained models for the experiments. Specifically, we integrated our classifier on 5 configurations of SC Resnet50 tested on the Imagenet Large Scale Visual Recognition Challenge (ILSVRC) 2012 dataset [31]. We selected the CRBQ method, which integrates an encoder-decoder architecture in the split point of the SC DNN. The last convolutional layer of the encoder (i.e., bottleneck) drastically reduces the number of channels of the output FM. The number of output channels defines the split configuration. In particular, among the SC configurations available in [30], we selected the configurations that in the corruption-free scenario reached an acceptable level of accuracy available in Table I (i.e., 70%): CRBQ(2), CRBQ(3), CRBQ(6), CRBQ(9), and CRBQ(12).

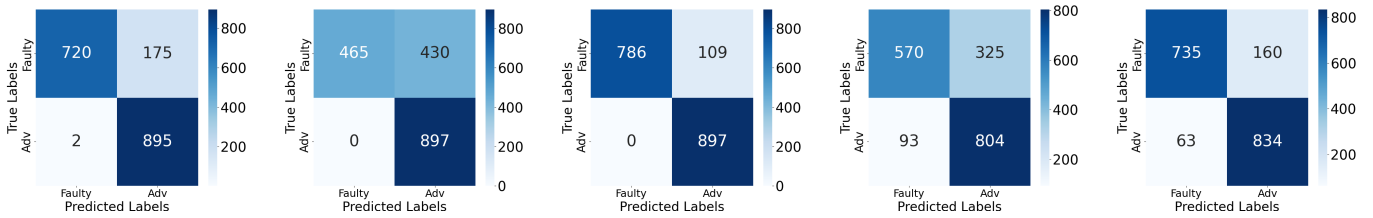
During the first step of our workflow, we conducted a series of campaigns to simulate *i)* hardware fault effect (through the FI campaigns) and *ii)* intentional corruption (through adversarial attack simulations) on the SC Head. To ensure consistency in the evaluation and generalization of the results, the same representative example of images (i.e., one for each target label) is selected from the Imagenet test dataset. In particular, Imagenet instances follow a hierarchical organization, enabling the grouping of the available classes into 50 instances. [31]. For this reason, we selected a representative sample of 50 images to perform our experiments. The encoder's output FMs were collected for each SC configuration under test to construct two datasets: the faulty dataset (FI-based corruptions) and the adversarial dataset (attack-based corruptions).

To conduct the FI campaigns, we utilized a modified version of PyTorchFI [22], which allows the corruption of DNN components (i.e., weights or neurons) during inference according to a user-defined function (e.g., bit-flip at a specific $bit\_location$). This version of PyTorchFI incorporates the error model outlined in Section III-A1. As the first step of 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 an FMA core. BlER was varied within the interval $[0.2, 1]$ to represent GPU architectures with differing numbers of SMs. 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 negligible impact [13]. For each configuration of the parameter vector (TS, $BlER$, NER, $bit\_location$), the inference was repeated five times, resulting in a total of 1,600 experiments per campaign. Similarly, to generate the adversarial dataset, we applied our attack methodology outlined in Section III, using the same 50 original examples. Each input was subjected to 300 attack iterations, producing 15,000 adversarially perturbed feature maps. We decided to stop after 300 iterations of the attack, as they are entitled to change the label on most images.

We performed two types of undersampling: the first to balance the two classes by reducing the number of faulty samples to match the number of adversarial samples, and the second applied to both classes to eliminate redundancies in the data, making the dataset more diverse and robust. This second step helped prevent the model from converging too quickly, improving generalization and allowing for a more detailed observation of the learning process over time. After undersampling, the complete dataset for each configuration was divided into three sets: 4,183 samples for training, 1791 samples for validation, and 1,792 samples for testing.

Finally, during our last step, we employed a trial-and-error approach to find the best hyperparameters, considering:

- *Learning rate*. Fully connected neural networks have many parameters, leading to a complex loss function landscape filled with many local minima. Using a high learning rate can hinder the model's ability to converge to an optimal solution, as it may cause the optimizer to overshoot critical areas of the loss function. For this reason, our search space has a learning rate that varies within the interval $[10^{-6}, 10^{-3}]$ [32].
- *Weight decay*. The interval $[10^{-2}, 10^{-4}]$ was chosen for weight decay tuning as it balances the trade-off between overfitting prevention and model capacity retention. Values closer to 10^{-2} provide stronger regularization, beneficial for small or redundant datasets, while values near 10^{-4} allow greater flexibility in learning complex patterns without excessive constraint. This range is commonly used in fully connected networks, ensuring an optimal balance between stability and generalization [32].
- *Dropout rate*. The interval $[0.3, 0.8]$ for the dropout rate



(a) CRBQ (2) (b) CRBQ (3) (c) CRBQ (6) (d) CRBQ (9) (e) CRBQ (12)

Fig. 3: Confusion Matrices for the CRBQ configurations.

was chosen to explore a range that effectively balances regularization and model performance. A dropout rate of 0.3 provides moderate regularization, while 0.8 offers a stronger regularization effect, reducing the risk of overfitting. This range is consistent with common practices in dropout regularization and ensures the model retains sufficient capacity for learning while preventing overfitting [32].

- *Hidden layers neurons*. We experimented with configurations varying the number of neurons in the two hidden layers from 16 to 128. This range was selected to explore the trade-off between model complexity and generalization capability. Smaller networks with fewer neurons (closer to 16) tend to have lower representational power but better generalization. In comparison, larger networks (up to 128 neurons) can capture more complex patterns but may require additional regularization to prevent overfitting.

The experiments were performed on the High-Performance Computing (HPC) system of Politecnico di Torino [33]. Specifically, on a six-node cluster with two Intel Xeon Scalable Processors Gold 6130 2.10 GHz 16 cores, and equipped with 6 NVIDIA Tesla V100 SXM2, 32 GB, and 5,120 CUDA cores.

V. Experimental Results

The five models trained on the different configurations show robust performances over the test set. Table II presents the classification performance of models trained using data from different SC configurations. These results highlight the overall effectiveness of the approach considered and its ability to achieve good accuracy (from 75.91% to 93.91%). In particular, CRBQ(6) reaches an accuracy of 93.91%, with precision and recall values exceeding 94%. This suggests that the approach can provide a highly reliable classification between faulty and adversarial samples. Also, other configurations, such as CRBQ(2) and CRBQ(12), perform well, with accuracy values of 90.12% and 87.45%, respectively. This further supports the idea that the method is robust across different data sources.

Figure 3 provides a detailed visualization of our classifiers' performance through confusion matrices, complementing the quantitative results presented in Table II.

A consistent pattern emerges across configurations: true positive rates for adversarial examples consistently exceed those for faulty examples. Similarly, false positives for adversarial examples are higher than for faulty examples in all configurations except CRBQ(2). This suggests an inherent bias in our model toward classifying inputs as adversarial examples.

TABLE II: Classifier Accuracy, Precision, and Recall metrics per SC configuration. Precision and Recall metrics are normalized by the number of occurrences for each class (*Faulty* and *Adv*).

SC Configuration	*Accuracy*	*Precision*	*Recall*
CRBQ(2)	90.12%	91.68%	90.11%
CRBQ(3)	75.91%	83.74%	75.91%
CRBQ(6)	93.91%	94.58%	93.91%
CRBQ(9)	76.67%	78.59%	76.65%
CRBQ(12)	87.45%	87.92%	87.45%

The cause of this bias warrants further investigation, particularly regarding the composition of our training datasets. A potential explanation is the limited number of images used to generate both adversarial and faulty samples, which may have prevented the models from developing robust generalization capabilities across diverse inputs. Future work should address this limitation by expanding the training dataset and ensuring a more balanced representation across classes.

VI. Conclusions and Future Work

This work presents a novel AI-based classification approach to differentiate the nature of the corruption, whether due to adversarial attacks or to hardware fault effects in SC systems. The classifier receives as input the Feature Maps extracted from the Head of the SC DNN after *i)* the simulation of adversarial attacks, and *ii)* hardware-aware Fault Injection Campaigns. Our classifier is trained to distinguish intentional corruptions (adversarial attacks) from unintentional corruptions (hardware physical defects) by processing the collected data. Our experimental evaluations demonstrate that classification performance varies across SC configurations, with the best-performing model achieving an accuracy of 93.91% for CRBQ(6). The notable performance differences observed in configurations such as CRBQ(3) and CRBQ(9) warrant further investigation. The future work will involve conducting a comprehensive statistical analysis of data distribution characteristics to distinguish between small input corruptions and Silent Data Corruptions. Next steps include exploring alternative classification and anomaly detection techniques to address performance variations. Additionally, the focus will be on improving model robustness and extending the test bench to include various Split Computing configurations with different compression mechanisms in the Head model.

REFERENCES

[1] 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.

[2] 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.

[3] M. Moore *et al.*, "International roadmap for devices and systems," *Accessed: Jan*, 2020.

[4] Y. He, G. Meng, K. Chen, X. Hu, and J. He, "Towards security threats of deep learning systems: A survey," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1743–1770, 2020.

[5] N. F. Schneidewind, "Reliability-security model," in *11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'06)*. IEEE, 2006, pp. 9–pp.

[6] A. Carelli, A. Vallero, and S. Di Carlo, "Performance monitor counters: Interplay between safety and security in complex cyber-physical systems," *IEEE Transactions on Device and Materials Reliability*, vol. 19, no. 1, pp. 73–83, 2019.

[7] A. Carelli, A. Vallero, and S. D. Carlo, "Shielding performance monitor counters: a double edged weapon for safety and security," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, 2018, pp. 269–274.

[8] Y.-S. Hsiao, Z. Wan, T. Jia, R. Ghosal, A. Mahmoud, A. Raychowdhury, D. Brooks, G.-Y. Wei, and V. J. Reddi, "Silent data corruption in robot operating system: A case for end-to-end system-level fault analysis using autonomous uavs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 4, pp. 1037–1050, 2023.

[9] 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 *2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)*. IEEE, 2023, pp. 1–4.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] V. Turco, A. Ruospo, E. Sanchez, and M. S. 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.

[18] C. Nanjunda, M. A. Haleem, and R. Chandramouli, "Robust encryption for secure image transmission over wireless channels," in *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, vol. 2. IEEE, 2005, pp. 1287–1291.

[19] M. Kooli, F. Kaddachi, G. Di Natale, and A. Bosio, "Cache-and register-aware system reliability evaluation based on data lifetime analysis," in *2016 IEEE 34th VLSI Test Symposium (VTS)*. IEEE, 2016, pp. 1–6.

[20] A. Benso and S. DiCarlo, "The art of fault injection," *Journal of Control Engineering and Applied Informatics*, vol. 13, no. 4, pp. 9–18, 2011.

[21] 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.

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

[23] 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 *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2024, pp. 1–7.

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

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

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] PoliTo, "Hpc@polito," http://www.hpc.polito.it, last visited on March 14, 2025.