Towards the Integration of Reliability and Security Mechanisms to Enhance the Fault Resilience of Neural Networks

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 22, 2021, accepted November 4, 2021, date of publication November 17, 2021,
date of current version November 30, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3129149

Towards the Integration of Reliability and Security Mechanisms to Enhance the Fault Resilience of Neural Networks

NIKOLAOS IOANNIS DELIGIANNIS[1], (Member, IEEE), RICCARDO CANTORO[1], (Member, IEEE), MATTEO SONZA REORDA[1], (Fellow, IEEE), MARCELLO TRAIOLA[2,3], (Member, IEEE), AND EMANUELE VALEA[4], (Member, IEEE)

[1]Department of Control and Computer Engineering, Politecnico di Torino, 10129 Torino, Italy
[2]CNRS, IRISA, Inria, University of Rennes, 35000 Rennes, France
[3]CNRS, UCBL, CPE Lyon, INL, UMR5270, INSA Lyon, ECL, University of Lyon, 69007 Lyon, France
[4]CEA-List, Université Grenoble Alpes, 38000 Grenoble, France

Corresponding author: Nikolaos Ioannis Deligiannis (nikolaos.deligiannis@polito.it)

This work was supported by the Computational Resources provided by HPC@PoliTO, a project of Academic Computing managed by the Department of Control and Computer Engineering of the Politecnico di Torino.

ABSTRACT Nowadays, many electronic systems store valuable Intellectual Property (IP) information inside Non-Volatile Memories (NVMs). Designers widely use encryption mechanisms to enhance the integrity of such IPs and protect them from any unauthorized access or modification. At the same time, often such IPs are critical from a reliability standpoint. Thus, dedicated techniques are employed to detect possible reliability threats (e.g., transient faults affecting the NVM content). The weights of a neural network (NN) model (e.g., integrated into an object detection system for autonomous driving or robotics) are typical examples of precious IP items. Indeed, NN weights often constitute proprietary data, stemming from an extensive and costly training process; moreover, their correctness is key for the NN to work reliably. In this article, we explore the capability of encryption mechanisms to ensure protection from both reliability threats. In particular, we assess, via extensive fault injection campaigns, the capability of different memory encryption schemes – usually used only for security purposes – to detect faults and thus, enhance the reliability of the system. Experimental results show that, by cleverly choosing the proper encryption scheme, it is possible to achieve very high fault detection rates (greater than 99%) with respect to Multiple Bit Upsets. The gathered results pave the way to the integration of reliability and security mechanisms to achieve better results with lower costs.

INDEX TERMS Artificial neural networks, convolutional neural networks, encryption, fault detection, fault injection, non-volatile memories, reliability, security.

I. INTRODUCTION

Information technology is a major aspect of modern society. Digital systems have become widespread, considerably changing the way people interact with computing machines. The design process and the architectures of electronic systems had evolved considerably since their emergence several decades ago. Nowadays, designers must consider several constraints, including those related to reliability, and follow standards such as DO-254 for avionics and ISO-26262 for automotive to meet certain criteria and thresholds. These constraints derive from the needs of safety-critical systems. Indeed, these systems must be able to detect a sufficiently high percentage of faulty conditions that could compromise their correct operation, thus avoiding incurring critical failures, which in turn could endanger human lives or cause large economic losses.

In the last years, the interest in security-oriented techniques to prevent possible attacks on such systems has been exponentially growing. These attacks aim to either change the behavior of the systems or extract private and/or precious information (*Intellectual Property* or *IP*) from them [1]. Some of these IP data items are stored into Non-Volatile Memories (NVMs) that are an attractive target for malicious

The associate editor coordinating the review of this manuscript and approving it for publication was Jun Wang.

users due to the persistence of the data [2]. Studies have been conducted to evaluate and mitigate the security threats for NVMs [3]. NVMs are also prone to faults caused by, for example, radiation effects. To harden the memories with respect to faults, designers typically adopt redundancy solutions (e.g., Error Correction Codes, or ECCs) to detect the occurrence of single- and multiple-bit errors and possibly correct some of them [4]–[6]. On the other side, when the memory content represents a valuable IP, designers protect it against possible attacks via encryption [7].

A prime example of a system where both safety and security play a crucial role is an autonomous system [8] that employs the Machine Learning (ML) technology [9], [10]. Machine learning is a widely adopted technology in various sectors such as healthcare [11], automotive [12]–[14] and aerospace [15]. In these scenarios, the weights of the ML model represent a valuable asset [16] for the system since they are strongly linked to the application's overall functionality. Furthermore, the weights result from a typically long (hence, expensive), non-intuitive training process of the model. As a result, the weights of an ML model represent a valuable IP for the system and are typically stored into NVMs that shall not be compromised or tampered with.

Currently, existing solutions to protect the NVM content are not designed to protect from faults and malicious attacks at once. In fact, reliability experts decide about the former ones, while security experts deal with the latter, often with limited interactions between the two groups. This work originates from the observation that encryption mechanisms may also offer some interesting fault detection capabilities since, in some cases, they tend to amplify the effect of faults, making them more evident, and thus detectable. Hence, studying the reliability features of different encryption solutions becomes attractive. This enables taking also reliability into account when selecting the most suitable encryption mechanism for a given system. Indeed, we believe that designers who need to adopt encryption mechanisms in their systems may benefit from the intrinsic fault-detection capability of such mechanisms. This, in turn, facilitates the achievement of the target reliability goals for the system.

In this work, we experimentally evaluate the positive effects that data encryption/decryption may have in terms of reliability enhancements with respect to the effects of possible transient faults. In particular, we focus on systems having NVMs already provided with encryption/decryption mechanisms to protect the stored data from malicious attacks. As case studies, we use an Artificial Neural Network (ANN) and a Convolutional Neural Network (CNN), whose weights represent the IP, encrypted and stored in the NVM. In our experiments, we inject faults in the encrypted weights and analyze their effects on the Neural Network behavior and the system fault detection capabilities with and without encryption. We perform various experiments with different ciphers and extensive fault injection campaigns to evaluate the effect of the encryption on the system fault detection capabilities. The gathered experimental results show that by carefully selecting a cryptographic algorithm, we can achieve a very high rate of fault detection, in particular with respect to Multiple Bit Upsets (MBUs). Hence, this work paves the way to a more clever selection of an encryption mechanism protecting the stored memory content with respect to malicious attacks, and providing a sufficiently high reliability degree.

The major contributions of this work are the following.

1) A thorough analysis of the fault tolerance capabilities of different operational modes of the *Advanced Encryption Standard (AES)*, widely employed for memory encryption.
2) An extensive experimental validation of AES fault tolerance capabilities on a meaningful case study, a Convolutional Neural Network (CNN). To the best of our knowledge, this is the first work studying the AES fault tolerance capabilities in the context of a CNN.

Experimental results show that a particular AES mode of operation – the Propagating Cipher Block Chaining (PCBC) with padding – allows achieving nearly 100% fault detection for both considered Neural Networks with respect to the Single Event Upset (SEU) faults model and the Multiple Bit Upset (MBU) faults model with multiplicity varying from 10 to 500.

In this article, we significantly extend the work presented in [17], introducing a completely new analytical study of the effects of different encryption/decryption mechanisms on the system's reliability. Moreover, experimental results are now extended to both an ANN and a CNN, thus better supporting the claim that security mechanisms, if suitably selected, can provide advantages from the reliability point of view.

The remainder of the article is organized as follows. Section II reports an overview on the state-of-the-art work on NVM reliability and provides a preliminary background on memory encryption. Section III illustrates the study that we conducted. Section IV details both case studies along with the experimental setup. Section V illustrates the obtained experimental results and, finally, Section VI draws the conclusions.

II. STATE-OF-THE-ART AND BACKGROUND

A. NVM RELIABILITY

NVMs (e.g., flash memories) are widely used as a storage medium for numerous devices since they are characterised by high performance with low power consumption and large storage density. They are used by designers in various business sectors, e.g., the mobile phone industry and the automotive industry. However, there is a notable difference between the two aforementioned domains. In the latter, the memory design criteria are strongly influenced by safety standards (e.g., ISO-26262) since the memory is meant to be used in safety-critical systems on the vehicle. Conversely, when realizing systems that can hardly endanger human lives (such as a mobile phone), the design constraints tend to be more relaxed. For example, the data retention rate in an embedded flash memory that is planned to be used in a car spans

from 10 to 20 years; in the case of a mobile phone, the stored data may last for a maximum of 5 years. Moreover, devices expected to be in the field for a long period of time are prone to error accumulation primarily due to the aging of the hardware components. Furthermore, designers have to consider that NVMs are prone to errors due to radiation effects [18], [19] and error accumulation. A recent study [20] reports an unexpected error explosion phenomenon in flash memories, where multiple errors occur in flash blocks over several operation cycles that exceed the ECCs detection and correction capabilities.

The reliability of NVMs has been extensively studied [21]. Also, numerous design methodologies, based on ECCs, have emerged over the years to enhance the resilience of NVMs to (soft and hard [22]) errors. As prominent examples, we can mention IBM's Chipkill used in combination with dynamic bit-steering [23] and Intel's Lockstep [24]. The security of NVMs has also been thoroughly studied [25], [26].

The aspect of the reliability and the tolerance of ML applications to faults has been also extensively studied in the past [27]–[29]. In [30], the authors present a novel methodology that exploits the relationship between input, parameters and output of CNNs in order to detect and correct bit errors. In [31], the authors perform an analysis on the reliability of CNNs and employ fault tolerance techniques to enhance their reliability on GPUs.

Typically, reliability and security are two aspects that are accounted for separately and independently. The main objective of our work is to pave the way to new approaches combining the two aspects. An approach considering both aspects has been proposed in [32], in the context of remote patient monitoring. In [32], a cooperative communication scheme is proposed to ensure reliability, along with a cryptography mechanism for privacy preservation. In this work, we aim instead at analyzing the inherent fault-detection features of a prominent and widely-used security mechanism, i.e., encryption. The goal is to assess whether the security mechanisms provided by the encryption are suitable also for fault-detection.

B. MEMORY ENCRYPTION

NVMs are particularly sensitive in terms of security. Their ability to permanently retain the stored information makes them easily exploitable by invasive attacks. Through chip decapsulation, an attacker can obtain direct access to the NVM surface and perform several kinds of attacks. One common threat is IP stealing. This is achieved by reading out the content of the NVM, which normally contains application code and data that represent a valuable IP for the company producing the target system. Another threat stems from the possibility for the attacker to tamper with the NVM content and provoke malfunctions in the processing elements that could ultimately result in privilege escalation on the system. A famous example is the code reuse attack and its variants, such as Return-oriented Programming (ROP) [33], [34] and Jump-oriented Programming (JOP) [35].

In this article, we focus on machine learning applications based on neural networks for safety-critical systems. In this context, the neural network weights stem from a long and expensive training process, making them a valuable asset. Moreover, in many safety-critical applications, the computing systems are deployed in close proximity to the user, making them easily accessible for the aforementioned physical attacks.

Memory encryption is a powerful mechanism for counteracting such threats. If the NVM content is fully encrypted, an attacker who obtains physical access to the memory cannot perform the aforementioned attacks. In fact, even if the attacker can read out the memory, the encryption makes the understanding of its content impossible. Moreover, encryption makes tampering-based attacks much more complex: an attacker would have to modify the encrypted data so that the decryption mechanism transforms them into data causing the desired corrupt behavior.

Validating the different solutions for the possible attacks has been a popular area of investigation. For example, in [36], the authors investigate the security aspect of a Deep Neural Network (DNN) application on a low-cost microcontroller. Specifically, they perform fault injections on the system while the DNN classifier is active and observe the role various activation functions have in terms of result misclassification. In [37], the authors explore the impact of bit-flip attacks on a wide variety of neural network architectures and showcase that it is possible for a hardware fault attack to dramatically lower the accuracy of the networks (up to 99%). Lastly, they propose heuristics for the identification of possible vulnerable parameters on such networks.

However, encryption alone is not capable of exhaustively detecting memory corruptions. More powerful techniques based on integrity primitives (e.g., authenticated encryption) can protect computing systems against most kinds of perturbations (i.e., fault attacks) that involve the memory content [38], [39]. In this article, we do not deal with artificial faults induced by an attacker, but we focus on natural faults coming from environmental sources instead. Although many similarities exist between artificial and natural faults, these are two problems that are traditionally dealt with very different technologies. In fact, protection mechanisms against fault attacks are based on security techniques, possibly based on cryptographic primitives (e.g., the already mentioned authenticated encryption), while natural faults are dealt with memory hardening techniques (e.g., error-correcting codes). This article introduces the possibility of dealing with natural faults relying on techniques belonging to the security domain. We consider very simple memory encryption techniques without the additional cost of the data integrity primitives, and we evaluate their properties in aiding the processing system in detecting natural faults that can affect the data stored in the NVM.

We consider a memory encryption implementation where data are loaded into the NVM already encrypted. When data are read by the processing element (in our case,

a general-purpose CPU), they are streamed through a hardware decryption module interposed between the NVM and the CPU. The encryption algorithm is based on a symmetric cryptography primitive, where the same secret key is used for both encryption and decryption. In the memory decryption scenario, the secret key must be stored inside the decryption module, possibly hardwired inside the module logic to avoid easy access through invasive attacks.

On the same architecture, we evaluate several encryption algorithms, all based on the Advanced Encryption Standard (AES), which can be implemented according to different *modes of operation*. The common element is a pseudorandom permutation (PRP) that processes a 128-bit block of plaintext to generate a 128-bit block of ciphertext. The PRP is conceived to have the following characteristics:

1) The permutation is dependent on the secret key. This implies that if the key is not known, the permutation looks like a random transformation. Hence, it is unfeasible to derive the corresponding plaintext only by knowing a ciphertext.
2) The permutation is invertible. This allows building the decryption function using the same key.
3) The permutation has *confusion* and *diffusion* properties. This means that each bit of the output is dependent on all the 128 bits of the input. For the fault detection purpose, this is a critical property because the corruption of one bit on the input block results in the corruption of the whole output block. In the following, we will refer to this property as *fault spreading*, that is related to the multiplicity and to the location of the errors stemming from a single bit error on the input message.

The AES basic PRP, also called *block cipher*, allows building several types of ciphers with different characteristics (i.e., different modes of operation).

III. ANALYSIS OF THE AES FAULT DETECTION CAPABILITY

This section details the thorough analysis that we performed on the fault detection capabilities of different encryption mechanisms. In particular, we define the different AES modes and highlight their capability to spread the fault effects.

A. AES MODES OF OPERATION AND THEIR FAULT SPREADING PROPERTY

In the following, we detail the modes of operation that we analyze in this article, highlighting their fault spreading properties:

- **Cipher Block Chaining (CBC) mode**: in this mode of operation, each block of plaintext is added to the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. In the decryption function, each plaintext block is added to the previous ciphertext after decryption. This implies that a 1-bit corruption on the ciphertext block i is propagated to the whole 128-bit plaintext block i, plus 1 bit of the plaintext block $i+1$. This is because the addition operation (i.e., a bit-wise XOR) does not spread the fault, but it simply transmits it to the corresponding bit of the result (Fig. 1a). Thus the fault spreading of the CBC mode is equal to 1 block plus 1 bit of the next block.
- **Cipher Feedback (CFB) mode**: in this mode of operation, each ciphertext block is computed as the sum of the corresponding plaintext block plus the encryption of the previous ciphertext block. In the decryption function, each plaintext block is computed as the sum of the corresponding ciphertext block and the encryption of the previous ciphertext block. Here, the encryption of the ciphertext block i is used as a keystream for both the encryption and the decryption of the block $i+1$. This implies that a 1-bit corruption on the ciphertext block i is transmitted to the corresponding bit of the plaintext block i, and it is also spread over the entire block $i+1$ (Fig. 1b). Thus the fault spreading of the CFB mode is equal to 1 bit in the present block plus the entire next block.
- **Propagating CBC (PCBC) mode**: in this mode of operation, each block of plaintext is added to both the previous plaintext block and the previous ciphertext block before being encrypted. This leads to a similar decryption behavior, i.e., each block of plaintext is added to both the previous plaintext block and the previous ciphertext block after the decryption function. This implies that a 1-bit corruption on the ciphertext block i is spread over the plaintext block i, plus all the following blocks up to the last one (Fig. 1c). Thus the fault spreading of the PCBC mode is equal to the number of blocks present between the block where the fault has happened and the last block of the encrypted data.
- **Counter (CTR) mode**: in this mode of operation, the encryption function is applied to a sequence of values that are generated by a counter initialized by a seed. The resulting output blocks (i.e., the keystream) are added to the plaintext blocks to obtain the ciphertext blocks. The decryption operation is performed by generating the same keystream and adding it to the ciphertext blocks in order to obtain the plaintext blocks. This implies that a 1-bit corruption on a ciphertext block is propagated to the same bit on the resulting plaintext block (Fig. 1d). Thus, the fault spreading of the CTR mode is equal to 1 bit in the same encrypted block.
- **Output Feedback (OFB) mode**: in this mode of operation, a keystream is generated starting from an initialization value that is passed through the encryption function multiple times. The ciphertext block is obtained as the sum between the plaintext block and the corresponding keystream block. In the decryption operation, the same keystream is generated (i.e., starting from the same initialization value), and this is added to the ciphertext blocks to compute the corresponding plaintext blocks.

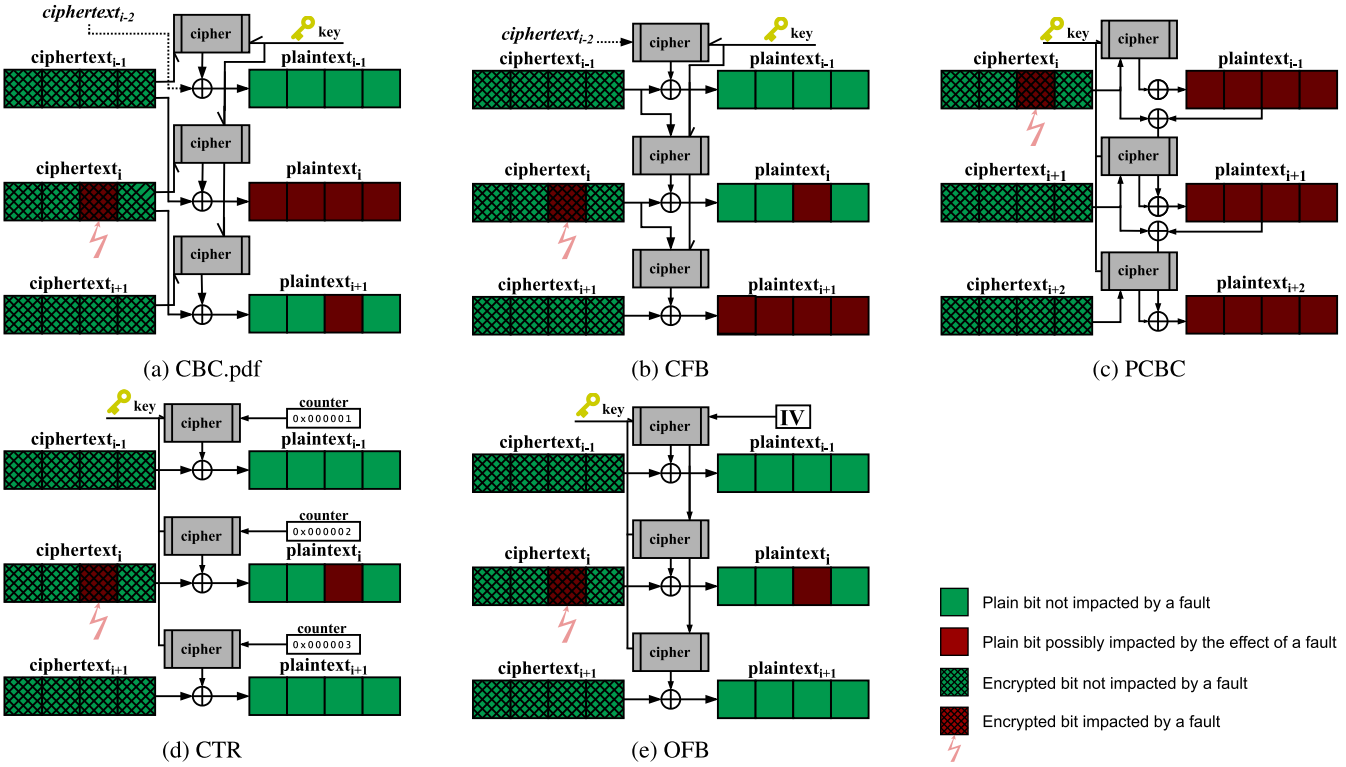

FIGURE 1. Decryption process of different AES modes of operations and their related fault spreading property.

This implies that a 1-bit corruption on a ciphertext block is propagated to the same bit on the resulting plaintext block (Fig. 1e). Thus the fault spreading of the OFB mode is equal to 1 bit in the same encrypted block.

From this point forward, we separate the AES modes of operation into two different categories, according to their fault spreading property:

- **Non-spreading category** including the CTR and the OFB modes of operation.
- **Spreading category** including the CBC, CFB and PCBC modes of operation.

In the *non-spreading* category, the faults on the ciphertext are not spread over the plaintext during decryption, but they are transmitted to the corresponding bit. Conversely, modes of operation in the *spreading* category can spread the effect of one-bit corruption on the ciphertext over at least an entire block of plaintext.

B. THE ROLE OF PADDING IN AES

Block-based encryption needs *padding* to work properly. Indeed, since the encryption is performed on 128-bit blocks, it is necessary to conceive a way to deal with plaintexts whose size is not multiple of 128 bits. The *padding standards* are conceived in order to add the number of bytes to the plaintext required to reach a multiple of 16 bytes (i.e., 128 bits). Thus, the size of the resulting ciphertext is always a multiple of 16 bytes. After decryption, the extra padding bytes are removed to obtain the original plaintext. The procedure used by the decryption module to determine the number of bytes that must be removed is mandated by the standard. One of the most popular padding techniques for block ciphers relies on the PKCS #7 - RFC 2315 standard [40]. According to this standard, if n bytes are added to pad the last block, then each of these bytes will encode the value n. After decryption, the last byte of the resulting plaintext is read, and its value determines the number of bytes that must be discarded. To provide an example, let us imagine a plaintext message of 100 bytes, which corresponds to 6 128-bit blocks plus 4 bytes. In order to complete the seventh block, 12 bytes are added as padding. Therefore, each padding byte will contain the value `0x0C` (12 in decimal). After decryption, the presence of the value `0x0C` on the last byte of the plaintext implies two things:

(i) the last 12 bytes of the resulting plaintext must all encode the value `0x0C`;
(ii) the last 12 bytes of the resulting plaintext must be removed after decryption.

C. FAULT TOLERANCE ANALYSIS

The phenomena described in Section III-A entail different consequences for the system.

The *non-spreading* modes of operation do not exacerbate the fault effect, leaving it confined to a specific bit of the plaintext. From a fault tolerance standpoint, a fault has the same probability of being masked or detected by the system or by the running application as if no encryption was applied.

Conversely, the *spreading* modes of operation aggravate the fault effect by extending it to multiple plaintext bits. From a fault tolerance standpoint, this could potentially entail worse consequences if no extra detection mechanism is available in the hardware platform or the running application.

Therefore, using the spreading AES modes of operation may seem extremely counterproductive, as it aggravates the fault effects and does not help detect their presence. However, as mentioned in Section III-B, the padding standard introduces some redundancy that may improve the fault tolerance. Indeed, storing the information about the number of added padding bytes into the padding bytes themselves (`0x0C` in the example in Section III-B) is not strictly necessary for the correct operation of the encryption. Nonetheless, the redundancy turns out to be a powerful property for fault detection. In fact, considering the above example, if the value `0x0C` appears on the last byte, but not all the 12 last bytes contain the same value, the decryption operation can detect and signal a decryption error.

Concerning the AES modes of operation described in Section III-A, for the padding check to be used for fault detection, the effect of a fault must reflect on the padding bits. In general, the event of having a fault impacting a padding bit is as probable as for the other bits. Thus, for non-spreading operation modes, the presence of the padding check is not likely to have a significant impact on fault tolerance. Conversely, when the fault effect is extended to other bits (as in spreading operation modes), the probability of obtaining a corrupted padding increases, and so does the detection capability. In detail, CBC and CFB (Figs. 1a and 1b) operation modes propagate the effect of a fault occurring on a single ciphertext bit only to the corresponding plaintext block and to the next one. Therefore, also these two operation modes are not much likely to benefit from the padding check. Nevertheless, the PCBC mode (Fig. 1c) has the interesting property to propagate a fault in a given block to all the successive blocks, all the way to the padding blocks. As a result, the probability of spreading the effect of a fault to the padding and detecting it is much higher in the PCBC mode of operation.

IV. EXPERIMENTAL VALIDATION OF AES FAULT DETECTION CAPABILITY

In this section, we describe the experimental setup that we adopted to validate the AES fault detection capabilities discussed in Section III. Firstly, we describe the adopted fault models and the classification that we use to categorize faults depending on their effects on the application under study. Then, we present the two ML applications used as case studies, and the adopted fault injection setup and experimental flow.

A. FAULT MODELS, FAULT CLASSIFICATION, AND FAULT EFFECTS

For the purpose of our analysis, in order to model the transient faults affecting the NVM that stores the IP of our system, we consider the *Single Event Upset (SEU)* (error multiplicity equal to 1) and the *Multiple Bit Upset (MBU)* (error multiplicity $>$ 1). We perform fault injection campaigns only on the ML application weights and not on other ML data or application code. Concerning the application code, in a previous work [41] we have shown that encryption enables high fault detection rates. We classify faults as follows:

- *Safe*: the fault does not impact the classification results of the ML application and is not detected. We consider 2 sub-categories of safe faults:
  1) *Masked*: the faulty results match the expected ones, i.e., the fault-free (golden) classification results. The fault does not propagate to the outputs of the network.
  2) *Critical Safe*: although the resulting classification matches the fault-free one, the fault reached the network outputs.

  The distinction between *Masked* and *Critical Safe* faults is only performed on ML applications that do not provide binary outputs (0/1).
- *Silent Data Corruption (SDC)*: the fault affects the classification results of the ML application and is not detected. The results do not match the golden classification results. The top-1 classification, namely the result predicted with the highest probability, is modified.
- *Detected*: we consider 2 fault detection mechanisms:
  1) *Exception*: the fault effect generates an 'illegal' condition and either the software or the hardware triggers an exception revealing the fault occurrence.
  2) *Decryption Detection*: the fault is detected by the decryption mechanism. In particular, the fault affects one (or more) of the padding bytes appended to the plaintext (weights) for the encryption; the decryption mechanism detects the padding incorrectness and triggers an error, allowing the detection of the fault occurrence.

It has been shown that ML-based systems are rather resilient to errors [42]–[44]. Obviously, we want to avoid *SDC* cases that may be catastrophic for the system and its environment. For example, in a self-driving vehicle, an object detection ML application impacted by a fault could lead to incorrect detection of, for instance, pedestrians. This could put human lives in harmful situations [45].

In most ML applications, the weights are represented by floating-point numbers. The IEEE-754 standard [46] specifies a special value, 'Not a Number' (NaN), resulting from invalid operations. The presence of a NaN value reveals an incorrect behavior and, in our scenario, triggers an exception. According to the IEEE-754 standard, a sequence of bits interpreted as NaN satisfies the following conditions:

1) the exponent bits are all set to 1,
2) at least one bit of the mantissa is set to 1.

NaN values may be detected at hardware level by the CPU or at software level by the application code. In both

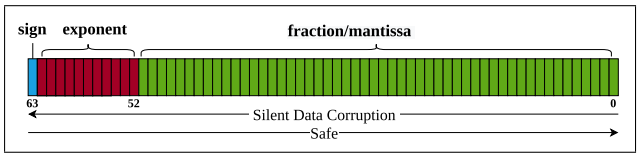

FIGURE 2. IEEE-754: Binary64 floating point number representation.

cases, an exception is typically triggered. In our context, a fault affecting an encrypted weight is detected if its effect generates either a NaN value – thus a software exception – or a corrupted padding caught by the decryption mechanism – thus a hardware exception. Otherwise, it will be either a masked fault or an SDC. This depends on the fault criticality, which is connected to the fault location. In Figure 2 we report the binary representation of a 64-bit floating-point number. If the effect of an undetected fault happens to corrupt one (or more) of the Least Significant Bits (LSBs) of the floating-point number, then it will most probably be a safe fault (critical safe or masked) since the change of the weight's value will not be significant. Conversely, as the fault location moves towards the Most Significant Bits (MSBs), the effects will be more severe and can lead to SDC [47], [48]. More in detail, as observed in [47], faults impacting the sign and the mantissa bits have a weaker impact on the network behavior than faults impacting the exponent bits.

B. CASE STUDY A: SIMPLE ANN

The first case study used for our experiments is an ANN. It is a classifier, which was developed using an ANSI C library [49]. Given as input a point in the (x, y) Cartesian plane, the ANN assigns it to one of the three following classes:

- C1: The point belongs to either one of the circles: $(x \pm 1)^2 + (y \pm 1)^2 \geq 0.16$
- C2: The point belongs to either one of the disks: $0.16 < (x \pm 1)^2 + (y \pm 1)^2 < 0.64$
- C3: The point belongs to neither circle nor disk: $(x \pm 1)^2 + (y \pm 1)^2 \geq 0.64$

The aforementioned loci are depicted in Figure 3. The training and the testing set of the network contain $3,000$ points each ($6,000$ in total). In each set $1,500$ randomly generated points are located inside the $[0, 2] \times [0, 2]$ rectangle and $1,500$ points are located inside the $[0, -2] \times [0, -2]$ rectangle.

The network is composed of 1 input layer, 3 hidden layers, and 1 output layer. The input layer has 2 neurons, one for each of the coordinates of the points (x, y). The hidden layers have 10 neurons each, and the output layer, which has 3 neurons, one for each of the classes (C1, C2, C3). The neurons of the output layer of the network provide a binary value (0/1). In total, the network contains 283 weights (including each neuron's BIAS input weight), each corresponding to a 64-bit floating-point number. To train the network, we used the supervised learning technique. Every point in our training and test dataset was encoded using one-hot encoding. The adopted training algorithm was *gradient descent*.

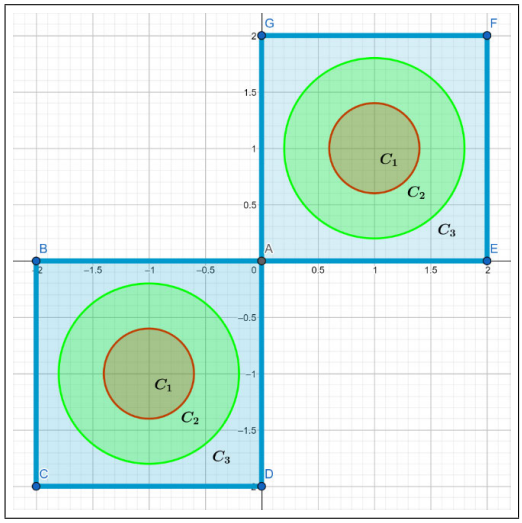

FIGURE 3. Graphical visualisation of the loci determined by the classification boundaries of the target ANN.

The generalization error of the network was found to be 1.33%. This is the probability for the classifier to misclassify a given point of the test-set (e.g., to classify a point of the class C1 as a point of the class C2 or C3). The activation function selected for this network is the *sigmoid function*.

C. CASE STUDY B: CNN

The CNN that we used for our experiments is the LeNet-5 network [52]. It was first introduced in [51], where it was used to detect handwritten zip codes digits [21].

We resort to a LeNet-5 variant [50] trained on the MNIST dataset of handwritten digits [53] using the darknet framework [54]. The CNN takes as input 28×28 pixel images, and its architecture, depicted in Figure 4, consists of the following layers:

- C1: A convolutional layer that produces as output 32 feature maps of size 28×28. C1 has 2,400 trainable weights.
- S2: A sub-sampling layer that reduces the dimension of the feature maps from 28×28 to 14×14. To generate a single value of a given output feature map, S2 takes the maximum value among a subset of four (2×2) input values.
- C3: A convolutional layer that produces 64 feature maps of size 14×14. C3 has 51,200 trainable weights.
- S4: A sub-sampling layer producing 64 feature maps of size 7×7, similarly to S2. Also S4 takes the maximum value among a subset of four (2×2) input values to generate an output value.
- FC5: A fully connected layer with 1024 neurons. FC5 has 3,211,264 trainable weights.
- FC6: A fully connected layer with 10 neurons. FC6 has 10,240 trainable weights.
- OUT At the output of the network, a *Softmax* operation is performed. This maps the output values to the range [0,1], to treat them as probabilities. Finally, the sum of

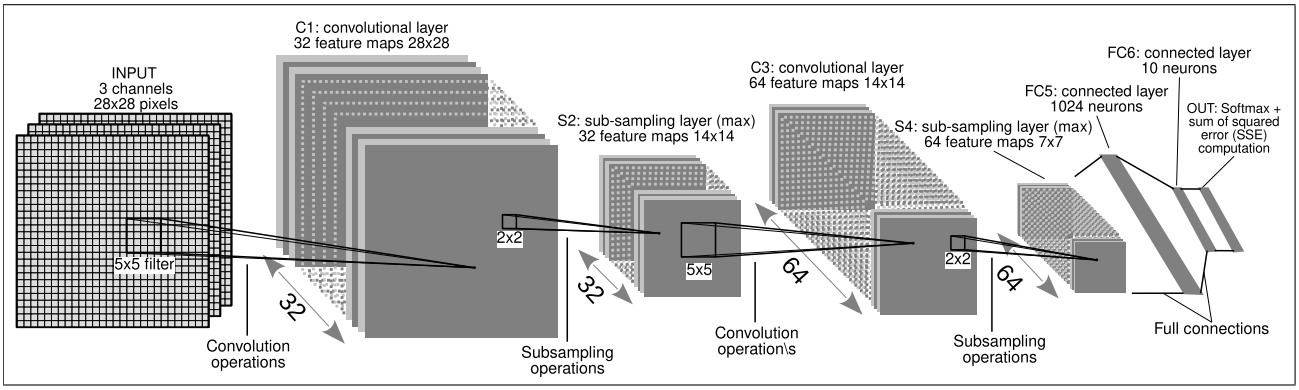

FIGURE 4. LeNet-5 variant architecture [50] (inspired from [51]).

TABLE 1. LeNet-5 classification accuracy, training and testing data for the MNIST dataset.

Digit	Accuracy	Number of Images	
		Training	Testing
0	95,87%	5,923	980
1	97,41%	6,742	1,135
2	87,39%	5,958	1,032
3	81.76%	6,131	1,010
4	94,42%	5,842	982
5	78,06%	5,421	892
6	91,45%	5,918	958
7	90,45%	6,265	1,028
8	89,07%	5,851	974
9	88,33%	5,949	1,009
		60,000	10,000

squared error (SSE) is calculated to compute the distance between the values from FC6 and some parameter vectors that correspond to the ten classes of digits. The parameter vectors were determined manually and kept fixed.

Neurons in layers C1, C3, FC5, and FC6 compute a dot product between their input vector and their weight vector and add a bias. For C1, C3, and FC5, the result is then passed through a *Rectified Linear Unit (ReLU)* activation function. For FC6, a linear activation function is used. Table 1 reports the percentage of images classified correctly per-digit after training the network along with the number of images that were used for the training and testing purposes of the network. In total, 70,000 images were used.

The total number of weights of the whole network, including neurons of both the convolutional and fully connected layers, is 3, 275, 104. For more details on the LeNet-5 structure and functionality, please refer to [51].

D. FAULT INJECTIONS

As already mentioned, in this study we focus on transient faults affecting the NVM that stores the ML application's weights. To correctly model this scenario, we performed fault injection campaigns on the encrypted version of the ML applications' weights, before decrypting and using them to execute the ML application. Table 2 presents the size (in terms of total number of bits used to represent the weights)

TABLE 2. Network sizes and total experiments.

Network	Total bits (net's weights)	Experiments	
		SEUs	MBUs
ANN	18,112	6	36
CNN	209,606,656		

of the two ML applications, along with the total amount of experiments performed.

For the experiments executed under the SEU fault model, we performed one fault injection campaign for each of the six considered cryptographic configurations. For the experiments executed under the MBU fault model, we performed six fault injection campaigns for every cryptographic configuration. Specifically, we injected faults of six different multiplicities, namely 10, 20, 50, 100, 200 and 500. Thus, the total amount of experiments (i.e., fault injection campaigns) related to the MBU model was $6 \times 6 = 36$.

1) FAULT INJECTIONS PER ML APPLICATION

In order to obtain statistically meaningful results with an error margin of ≈1.5% and a confidence level of 95% we had to perform 3, 454 fault injections for every experiment on the ANN and 4, 145 fault injections for every experiment on the CNN application. The number of injected faults per experiment was calculated according to [55] as:

$$fault\_injections = \frac{N}{1 + e^2 \times \frac{N-1}{t^2 \times 0.25}}$$

where:

- N is the population size, i.e., column 2 of Table 2.
- e is the desired error margin.
- t depends on the desired confidence level (t=1.96 corresponds to 95% confidence level).

Furthermore, a uniform distribution was used for each fault injection campaign. Hence, each memory bit had the same probability of being selected.

E. EXPERIMENTAL FLOW

Figure 5 depicts the flow of our experiments. To study the effect of the padding check on fault detection capabilities,

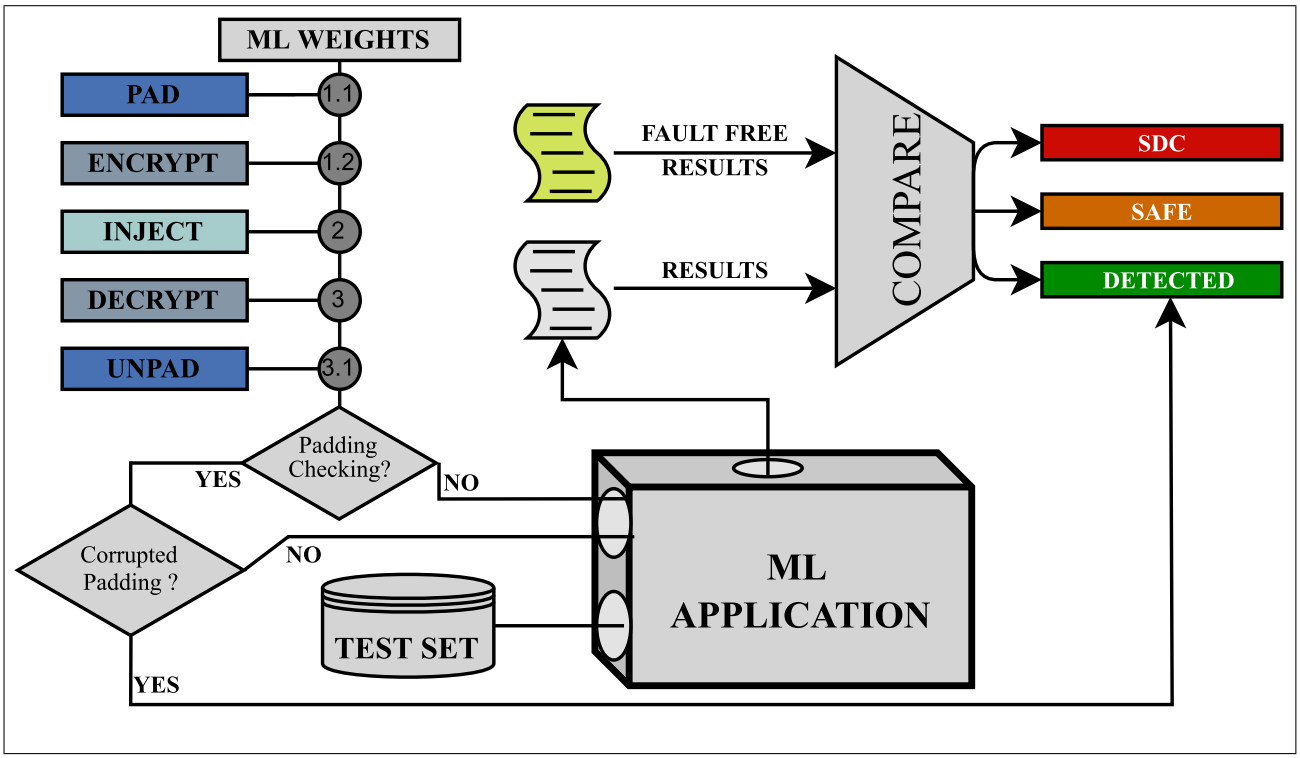

FIGURE 5. Experimental setup.

we organized the flow as follows. Firstly, the weights are either padded and then encrypted; then, a fault is injected in one of the weights, and finally they are decrypted and then un-padded. The effect of a fault on the encrypted memory content may propagate to the padding, thanks to the decryption. In this case, if padding integrity checks are performed during un-padding, the check mechanism detects that the padding bytes have been altered, thus leading to the fault detection. On the other hand, if a padding byte has not been altered (or if the target system does not perform the padding integrity check), then it is up to the application to possibly detect the fault. As already mentioned, in our scenario this happens only if a NaN is generated and a software exception is triggered. Finally, the classification results are compared with the golden classification results. Classification is performed as follows:

- If the results match i.e., exactly the same classification was performed with respect to the fault-free scenario, then the fault is classified as safe.
- If the results do not match i.e., items have been miss-classified, then the fault is classified as SDC.
- If the decryption mechanism detects a discrepancy in the padding segment or the application detects a NaN value while loading the weights, an exception is triggered and the fault is classified as detected.

In order to support the fault injection experiments we developed a tool using Python. This tool is responsible of (i) encrypting the weights of the respective ML application using a given cipher configuration, (ii) injecting a fault of a given multiplicity in the form of bit-flips and finally (iii) decrypting the memory data segment.

First of all, the tool executes the fault-free (golden) ML application with the given test set and obtains the fault-free NN results. Then, in order to have a point of reference and comparison, we perform fault injections on the networks' weights without using encryption. The criticality of the fault strongly depends on the fault location, as explained in Section IV-A. In this scenario, the only case where a fault is detected is when it causes a NaN value, which in turn triggers a software exception.

V. RESULTS

In this section, we present the experimental results that we obtained for our case studies. SEU results are summarized in Table 3, while MBU results are presented in the plots of Figure 6 and Figure 7. In both cases, two scenarios are considered. In the first scenario (NO PAD), checks on the padding segment of the ciphertext are not performed, while in the second scenario (PAD), checks are performed during the decryption process.

A. SEU EXPERIMENTS

1) ANN

Regarding the SEU experiments on the ANN, in the upper part of Table 3 we show that the differences between the results of the experiments performed with no encryption (our

TABLE 3. SEU results.

Fault Classification	NO ENC	ANN NO PAD *non-spreading* CTR	ANN NO PAD *non-spreading* OFB	ANN NO PAD *spreading* CBC	ANN NO PAD *spreading* CFB	ANN NO PAD *spreading* PCBC	ANN PAD *non-spreading* CTR	ANN PAD *non-spreading* OFB	ANN PAD *spreading* CBC	ANN PAD *spreading* CFB	ANN PAD *spreading* PCBC
Safe	75,5%	78,8%	77,4%	1,6%	2,6%	6,4%	76,6%	76,6%	0,6%	1,8%	0%
SDC	24%	21,8%	22%	98%	97,4%	92,8%	22,8%	22%	98,2%	96,4	0,6%
Detected	0,3%	0%	0,6%	0,4%	0%	0,8%	0,6%	1,4%	1,2%	1,8%	**99,4%**

Fault Classification	NO ENC	CNN NO PAD *non-spreading* CTR	CNN NO PAD *non-spreading* OFB	CNN NO PAD *spreading* CBC	CNN NO PAD *spreading* CFB	CNN NO PAD *spreading* PCBC	CNN PAD *non-spreading* CTR	CNN PAD *non-spreading* OFB	CNN PAD *spreading* CBC	CNN PAD *spreading* CFB	CNN PAD *spreading* PCBC
Safe	97,3%	96,8%	96,9%	20,3%	20,4%	0,0%	96,8%	96,9%	20,3%	20,4%	0,0%
SDC	2,7%	3,2%	3,1%	79,0%	78,6%	24,5%	3,2%	3,1%	79,0%	78 ,6%	0,1%
Detected	0,0%	0,0%	0,0%	0,7%	1,0%	75,5%	0,0%	0,0%	0,7%	1,0%	**99,9%**

NO PAD: Experiments that do not consider padding checking during decryption
PAD: Experiments that consider padding checking during decryption
The confidence level is 95% and the error margin is ≈ 1.5% (see Section IV-D)
-Total faults injected in the ANN: 3,454
-Total faults injected in the CNN: 4,145

reference baseline) and those performed with non-spreading encryption ciphers are not significant, regardless of the padding utilization. Thus, non-spreading encryption ciphers do not provide any enhanced fault-detection capabilities. The majority of the injected faults are classified as safe. Specifically, all these safe faults fall under the masked category since the network provides only binary outputs.

As for the spreading encryption ciphers, we observe that almost always, they produce more SDCs than the reference scenario, regardless of the padding utilization. The reason behind this behavior is the propagation of the fault during decryption. As explained in Section II-B, these modes of operation tend to amplify the fault effect by propagating it to neighbouring blocks of information. This attribute of the ciphers increases the probability of corrupting significant information bits that will eventually lead to an SDC case. However, the PCBC cipher configuration with padding utilization (PAD scenario) stands out for achieving a fault detection rate of 99,4%. This result is due to the nature of the PCBC decryption process: when a fault is injected in the ciphertext, the PCBC decryption mechanism propagates the fault effects to all of the following blocks, resulting in the corruption of the padding segment. Hence, the corruption is detected by the padding check, and an exception is raised.

2) CNN

The SEU results of the CNN case study are reported in the lower half of Table 3. Encryption with non-spreading ciphers does not provide any notable fault detection capabilities, as for the ANN scenario. Indeed, the results do not substantially deviate from the *no encryption* scenario, and the vast majority of the faults fall into the safe category. For the CNN, the output layer performs the softmax operation which maps the output values to the range [0,1]. Thus, the number of safe faults in this case is the sum of the masked and the critical safe faults (see Section V-C for further details). In [56], the authors, while considering LeNet-5 as a case study perform extensive fault injection campaigns on the network layers and classify the fault effects in a similar manner. We observe a similar behavior when comparing with the amount of faults classified as safe and SDC for the no encryption scenario of Table 3, thus confirming previous findings. As for the ANN results, CBC and CFB block ciphers produce many SDCs, regardless of the padding utilization. On the other hand, the PCBC configuration shows improved detection capabilities. In fact, when the padding check is not used, the PCBC can detect 75,5% of the faults. Moreover, when the padding check is performed, the PCBC achieves a detection rate of 99,9%.

One notable difference between the two case studies can be observed for the PCBC case in the NO PAD scenario. Without padding checks, PCBC achieves a higher fault detection rate when applied to the CNN, compared to the ANN. Note that the only way for a fault to be detected in this case is for the fault to generate a NaN value that will trigger an exception. We think that the reason behind this peculiarity may be the difference in size and number of operations between the two networks in combination with the PCBC's fault spreading property to all the following blocks. Indeed, the CNN's weights, as shown in column 2 of Table 2, are 10^4 times as many as the weights of the ANN network. In order to be encrypted with a spreading cipher configuration, the weights are split into 128-bit blocks. In the ANN there are 142 blocks whilst in the CNN there are 6,550,208 blocks. Thus, during the decryption process, the effect of a fault impacting a random encrypted block in the CNN will be propagated to much more subsequent blocks than in the ANN, thus impacting more weights. Moreover, much more operations are carried out in the CNN than in the ANN. This surely contributes to error accumulation and propagation and increases the probability of getting a NaN. To give an example, we think that when a fault impacts a lot of weights (thanks to PCBC spreading property), it is likely that one or more of them assume a large value. The large values would

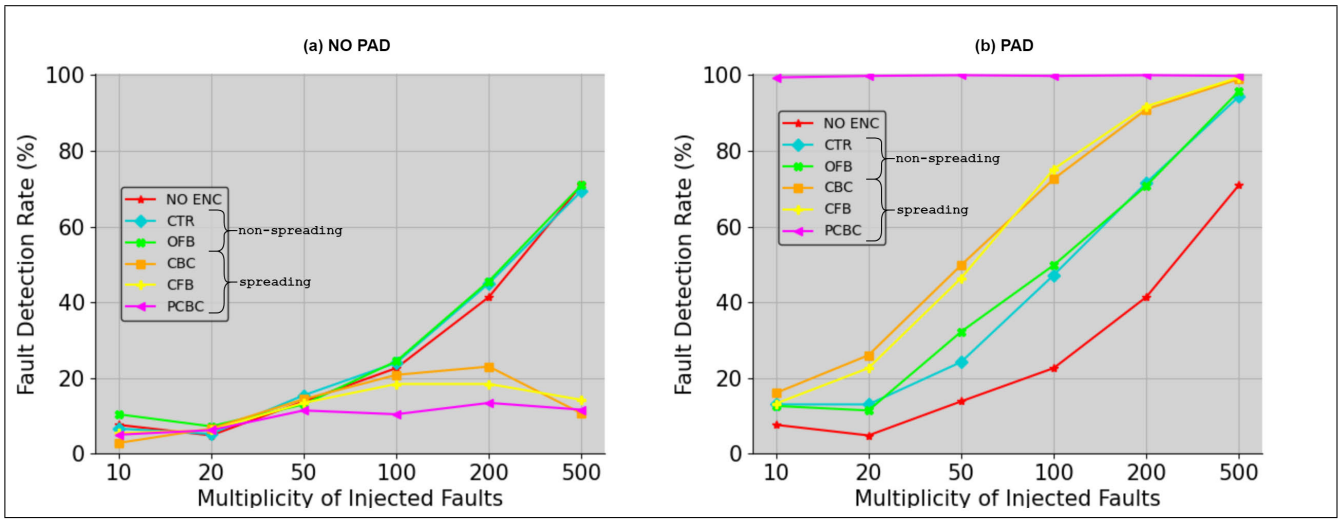

FIGURE 6. ANN MBU results for the (a) NO PAD and (b) PAD scenarios.

grow even bigger thanks to the convolution operations, which involve multiplications and additions, eventually turning into the infinity value. Ultimately, operations with infinity values (e.g., multiplication with zero or the $\infty - \infty$ operation) could more likely generate a NaN.

B. MBU Experiments

1) ANN

Figure 6 reports the results of the MBU experiments for the ANN application. Regarding the NO PAD scenario (Figure 6-a), we observe that non-spreading ciphers behave similarly to the no encryption scenario. As the fault multiplicity rises, the fault detection rates rise as well. This means that, as the number of injected faults rises, the probability of inducing a NaN increases. On the other hand, spreading cipher configurations tend not to provide any significant fault detection rate. Specifically, we can see that their fault detection rate drops for fault multiplicity higher than 200.

Concerning the PAD scenario where the padding integrity checks are performed (Figure 6-b), we observe PCBC dominating over the rest of the ciphers. Indeed, the PCBC achieved high fault detection capabilities, very close to 100%, regardless of the injected fault multiplicity. In general, the performance of all the ciphers in terms of their fault-detection capabilities was also enhanced since, in this scenario, a fault may corrupt bytes of the padding segment, generating an immediate detection.

2) CNN

The results of the CNN case study are depicted in Figure 7. Similarly to what happens in the SEU experiments, we can see that the plots deviate from the ANN case. Regarding the NO PAD scenario (Figure 7-a), non-spreading ciphers do not provide significant fault detection. In fact, they achieve a detection rate close to the no encryption scenario. On the other hand, the fault detection rates obtained for spreading ciphers are higher for the CNN than for the ANN. In particular, the PCBC configuration also showed high detection capabilities, very close to 100%. We think that this phenomenon is related to what was previously stated in Section V-A for the SEU experiment for the NO PAD scenario: the number of weights and operations in the CNN is much larger than in the ANN. Therefore, we think that, when multiple faults impact the CNN weights and the effect is spread to other weights thanks to spreading AES configurations, it is highly likely to generate high values that could eventually turn into infinity. Operations with infinity values could likely generate a NaN.

In the PAD scenario (Figure 7-b), where padding checks are performed, we observe again PCBC dominating over the rest of the ciphers by achieving very high fault detection rates, close to 100%.

3) SAFE FAULTS AND SDCs

As already discussed, the faults that remain undetected can be classified either as safe or SDC. In both the analyzed ML applications, the percentage of undetected faults classified as SDCs is directly proportional to the injected fault multiplicity. Consequently, the percentage of undetected faults that are classified as safe is inversely proportional to the fault multiplicity. Indeed, as the multiplicity of faults increases, the likeliness of an undetected fault impacting significant memory bits increases as well; thus, the probability of a fault being safe and not causing data corruptions decreases. We observed the same trend for both spreading and non-spreading cipher configurations.

While the trend is the same, the non-spreading configurations tend to produce higher percentages of safe faults (thus lower percentages of SDCs) than the spreading configurations for low fault multiplicities. This is due to the intrinsic property of these latter configurations to spread the fault effects to multiple bits in the decryption process.

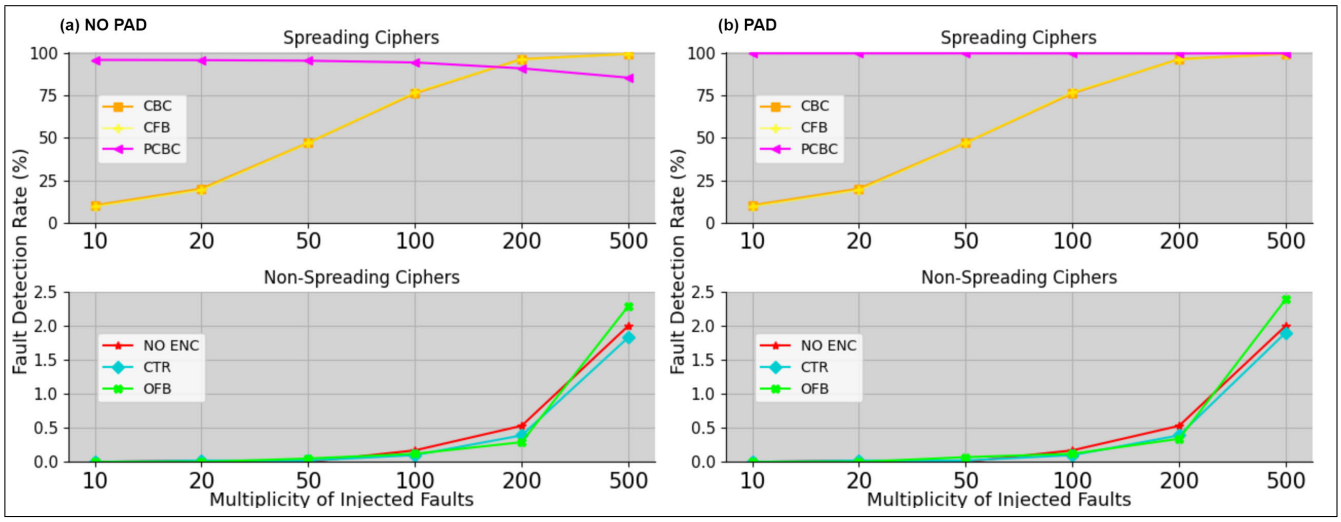

FIGURE 7. CNN MBU results for the (a) NO PAD and (b) PAD scenarios.

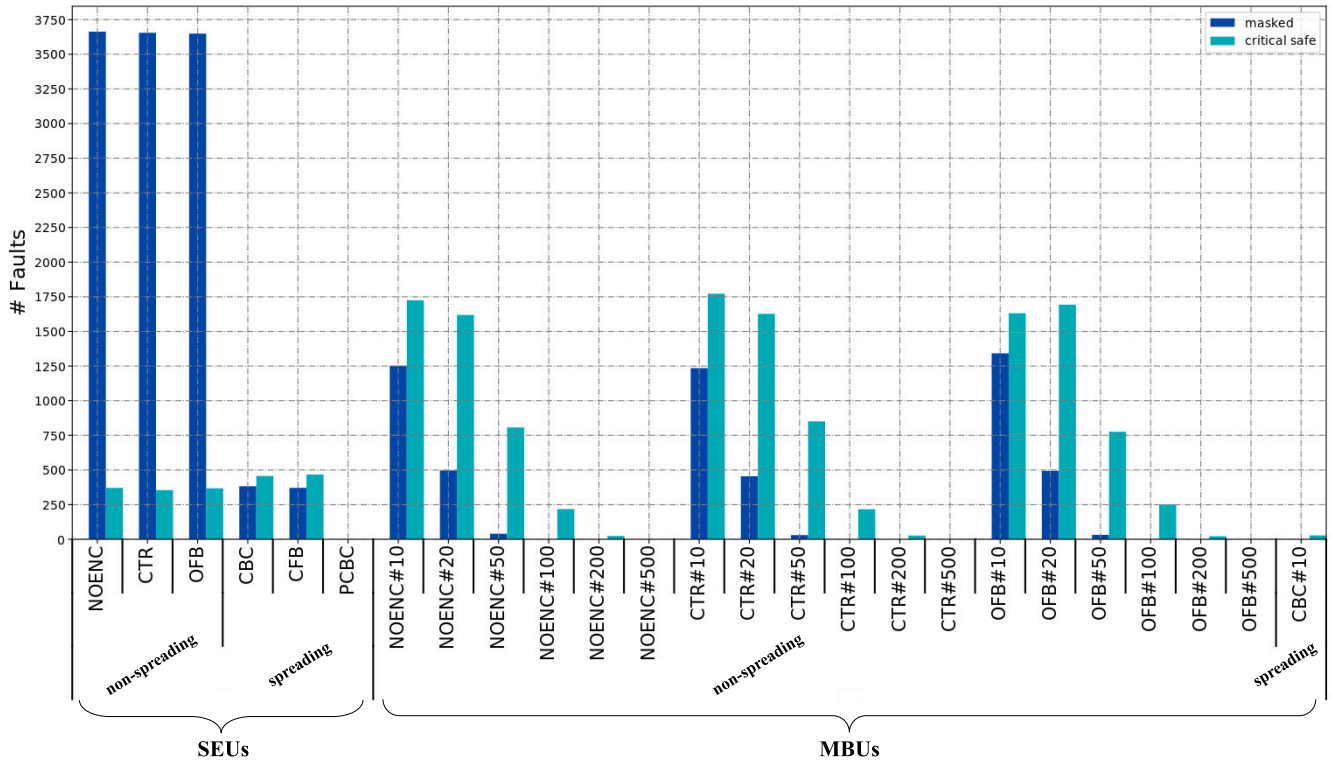

FIGURE 8. Analysis of safe faults for the CNN per fault injection campaign.

C. CNN: ANALYSIS OF SAFE FAULTS

In Section IV-A we defined safe faults as those faults that do not alter the functionality of the network. Namely, the classification produced by the faulty network is exactly the same as the classification produced by the fault-free network. However, for the case of the CNN, the network outputs are not binary, namely they map the output values to the range [0,1] via the softmax operation. Thus, we can further differentiate between masked faults and critical safe faults. A similar classification and analysis of the masked faults in CNNs is performed in [56], [57].

Figure 8 shows the total amount of the safe faults – divided into masked and critical safe – for each fault injection campaign performed on the CNN.

Firstly, let us consider the SEU fault injection campaigns. For the non-spreading cipher configurations, we observe that the vast majority of safe faults falls into the masked category. For the spreading cipher configurations we observe fewer cases of safe faults in general, since the vast majority of faults are classified as SDCs. Furthermore, the percentage of critical safe faults is higher than the masked ones for the spreading cipher configurations. Regarding PCBC, the amount of safe faults is zero. This behavior can be fully justified by the fault spreading properties of the employed ciphers as explained in Section III-A. In particular, there is a small chance that a non-spreading cipher configuration amplifies and propagate the fault effect, since one bit-flip on the ciphertext is responsible for one bit-flip on the plaintext. On the other hand, when it comes to spreading cipher configurations, the impact of a single fault is amplified, since a single bit-flip on the ciphertext corresponds to more than one bit-flip on the plaintext. More details are provided in Section III-A.

For the case of the MBU fault injection campaigns, on the x-axis of the bar plot, the multiplicity of each fault is noted after the name of the employed cipher configuration. Concerning the non-spreading cipher configurations, we observe a similar trend among the different configurations. In general, we observe a lower amount of safe faults w.r.t. the respective configuration in the SEU fault injection campaign. Then, we observe that the amount of critical safe faults is higher than the masked faults, while for the SEU fault injection campaign we had a number of masked faults higher than the number of critical safe faults. This is due to the higher fault multiplicities in the MBU fault injection campaigns: the higher the fault multiplicity, the higher the probability that the effect of a fault reaches the network outputs. Concerning the spreading cipher configurations, for almost every MBU fault injection campaign, no safe faults were identified and thus they are not included in the Figure 8. In the scenario of the spreading ciphers, each bit-flip performed is amplified during the decryption, since it corrupts the whole data block in which it is contained. Thus, even with the lowest multiplicity considered, namely 10, we observed a high amplification effect leading to either an SDC or a Detection. A very small amount of critical safe faults is only observed for the CBC configuration for the fault injection campaign with fault multiplicity equal to 10.

VI. CONCLUSION

Modern society is permeated with digital computing systems, which are increasingly vital to our everyday life. The design process of these systems has become incredibly complex, as many requirements have to be considered. In particular, reliability constraints have profoundly impacted the way designers implement these systems. Furthermore, in the last years, the growing interest in effectively facing malicious attacks on intellectual properties information within these systems led designers to adopt security-oriented techniques, such as memory encryption.

Autonomous systems employing Machine Learning (ML) technology are a prominent example where both reliability and security constraints are crucial. In particular, the correctness of the ML model weights determines the proper behavior of the system; simultaneously, the weights are also considered a precious Intellectual Property (IP) item since they result from an expensive and not trivial training process. Thus, companies need to protect them at once from faults and malicious attacks. Unfortunately, these two aspects are studied and handled separately, with little interaction between the respective experts.

In this work, we analyzed and highlighted the fault-detection capabilities offered by memory encryption mechanisms. The results of this work enable designers to single out the most suitable memory encryption mechanism for a system while taking into account not only its security but also its reliability. We experimentally evaluated the positive impact that data encryption has in terms of reliability enhancements with respect to the effects of transient faults. To do so, we performed extensive fault injection campaigns on the encrypted weights of an Artificial Neural Network (ANN) and of a Convolutional Neural Network (CNN) and evaluated the fault-detection capabilities provided by the decryption mechanism. The underlying idea is that the effect of a fault affecting encrypted data will spread to adjacent data in the decryption process, thus increasing the probability of detecting the fault occurrence. The obtained results show that selecting a particular Advanced Encryption Standard (AES) configuration, i.e., the Propagating Cipher Block Chaining (PCBC), in combination with padding check mechanisms allow to achieve very high fault detection rates ($> 99\%$), with respect to the Single Event Upset (SEU) and the Multiple Bit Upset (MBU) fault models.

This showcased behavior could be observed even in larger (i.e., deeper) networks than the CNN considered in our work. The fault spreading property of the PCBC scheme is independent of the plaintext size, which corresponds to the number of weights of the ML application in the considered scenario. Given that an appropriate padding checking mechanism is also employed, the overall fault detection capability of the system would be similar to the CNN case study analyzed in this article.

This work encourages and paves the way to developing new integrated design techniques that take into account multiple crucial requirements of new-generation advanced computing systems.

REFERENCES

[1] K. F. Li and N. Attarmoghaddam, "Challenges and methodologies of hardware security," in *Proc. IEEE 32nd Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, May 2018, pp. 928–933.

[2] S. Ghosh, M. N. I. Khan, A. De, and J.-W. Jang, "Security and privacy threats to on-chip non-volatile memories and countermeasures," in *Proc. 35th Int. Conf. Comput.-Aided Design*, Nov. 2016, pp. 1–6.

[3] M. N. I. Khan and S. Ghosh, "Assuring security and reliability of emerging non-volatile memories," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020, pp. 1–10.

[4] D. Rossi and C. Metra, "Error correcting strategy for high speed and high density reliable flash memories," *J. Electron. Test.*, vol. 19, no. 5, pp. 511–521, Oct. 2003.

[5] W. Liu, J. Rho, and W. Sung, "Low-power high-throughput BCH error correction VLSI design for multi-level cell NAND flash memories," in *Proc. IEEE Workshop Signal Process. Syst. Design Implement.*, Oct. 2006, pp. 303–308.

[6] J. Xiao-Bo, T. Xue-Qing, and H. Wei-Pei, "Novel ECC structure and evaluation method for NAND flash memory," in *Proc. IEEE Int. Syst.-Chip Conf. (SOCC)*, Sep. 2015, pp. 100–104.

[7] M. Ye, K. Zubair, A. Mohaisen, and A. Awad, "Towards low-cost mechanisms to enable restoration of encrypted non-volatile memories," *IEEE Trans. Depend. Sec. Comput.*, vol. 18, no. 4, pp. 1850–1867, Jul. 2021.

[8] S. Katzenbeisser, I. Polian, F. Regazzoni, and M. Stöttinger, "Security in autonomous systems," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2019, pp. 1–8.

[9] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[11] M. K. K. Leung, H. Y. Xiong, L. J. Lee, and B. J. Frey, "Deep learning of the tissue-regulated splicing code," *Bioinformatics*, vol. 30, no. 12, pp. i121–i129, Jun. 2014.

[12] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun, "Pedestrian detection with unsupervised multi-stage feature learning," Apr. 2013, *arXiv:1212.0142*.

[13] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1915–1929, Aug. 2013.

[14] C. Garcia and M. Delakis, "Convolutional face finder: A neural architecture for fast and robust face detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 11, pp. 1408–1423, Nov. 2004.

[15] W. R. Daniel and W. W. Andrew, "Applying machine learning-based diagnostic functions to rotorcraft safety," in *Proc. 17th Austral. Int. Aerosp. Congr. (AIAC)*, Feb. 2017, p. 663.

[16] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction Apis," in *Proc. 25th USENIX Conf. Secur. Symp.*, Berkeley, CA, USA: USENIX Association, 2016, pp. 601–618.

[17] R. Cantoro, N. I. Deligiannis, M. S. Reorda, M. Traiola, and E. Valea, "Evaluating data encryption effects on the resilience of an artificial neural network," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2020, pp. 1–4.

[18] C. Andreani, R. Senesi, A. Paccagnella, M. Bagatin, S. Gerardin, C. Cazzaniga, C. D. Frost, P. Picozza, G. Gorini, R. Mancini, and M. Sarno, "Fast neutron irradiation tests of flash memories used in space environment at the ISIS spallation neutron source," *AIP Adv.*, vol. 8, no. 2, Feb. 2018, Art. no. 025013.

[19] M. Bagatin, S. Gerardin, A. Paccagnella, A. Visconti, L. Chiavarone, M. Calabrese, and C. D. Frost, "Sensitivity of NOR flash memories to wide-energy spectrum neutrons during accelerated tests," in *Proc. IEEE Int. Rel. Phys. Symp.*, Jun. 2014, p. 3.

[20] Y. Pan, H. Zhang, M. Gong, and Z. Liu, "Unexpected error explosion in NAND flash memory: Observations and prediction scheme," in *Proc. IEEE 29th Asian Test Symp. (ATS)*, Nov. 2020, pp. 1–6.

[21] D. Ielmini, A. S. Spinelli, and A. L. Lacaita, "Recent developments on flash memory reliability," *Microelectron. Eng.*, vol. 80, pp. 321–328, Jun. 2005.

[22] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, "FREE-p: Protecting non-volatile memory against both hard and soft errors," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, Feb. 2011, pp. 466–477.

[23] T. J. Dell, "A white paper on the benefits of chipkill-correct ECC for PC server main memory," IBM Microelectron. Division, Armonk, NY, USA, Tech. Rep., 1997.

[24] Intel. *Independent Channel Vs. Lockstep Mode Drive Your Memory Faster or Safer*. Accessed: Nov. 5, 2020. [Online]. Available: https://software.intel.com/content/www/us/en/develop/blogs/independent%-channel-vs-lockstep-mode-drive-you-memory-faster-or-safer.html

[25] S. Liu, A. Kolli, J. Ren, and S. Khan, "Crash consistency in encrypted non-volatile main memory systems," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 310–323.

[26] A. Awad, M. Ye, Y. Solihin, L. Njilla, and K. A. Zubair, "Triad-NVM: Persistency for integrity-protected and encrypted non-volatile memories," in *Proc. 46th Int. Symp. Comput. Archit.*, Jun. 2019, pp. 104–115.

[27] S. Bettola and V. Piuri, "High performance fault-tolerant digital neural networks," *IEEE Trans. Comput.*, vol. 47, no. 3, pp. 357–363, Mar. 1998.

[28] V. Piuri, "Analysis of fault tolerance in artificial neural networks," *J. Parallel Distrib. Comput.*, vol. 61, no. 1, pp. 18–48, Jan. 2001.

[29] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[30] J. Ponader, K. Thomas, S. Kundu, and Y. Solihin, "MILR: Mathematically induced layer recovery for plaintext space error correction of CNNs," in *Proc. 51st Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2021, pp. 75–87.

[31] F. F. dos Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, "Analyzing and increasing the reliability of convolutional neural networks on GPUs," *IEEE Trans. Rel.*, vol. 68, no. 2, pp. 663–677, Jun. 2019.

[32] G. Mehmood, M. Z. Khan, A. Waheed, M. Zareei, and E. M. Mohamed, "A trust-based energy-efficient and reliable communication scheme (Trust-based ERCS) for remote patient monitoring in wireless body area networks," *IEEE Access*, vol. 8, pp. 131397–131413, 2020.

[33] S. Hovav. (2008). *Return-Oriented Programming: Exploits Without Code Injection*. [Online]. Available: https://www.blackhat.com/html/bh-usa-08/bh-usa-08-archive.html

[34] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-oriented programming: Systems, languages, and applications," *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 1, pp. 1–34, Mar. 2012.

[35] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang, "Jump-oriented programming: A new class of code-reuse attack," in *Proc. 6th ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS)*, 2011, pp. 30–40.

[36] X. Hou, J. Breier, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Security evaluation of deep neural network resistance against laser fault injection," in *Proc. IEEE Int. Symp. Phys. Failure Anal. Integr. Circuits (IPFA)*, Jul. 2020, pp. 1–6.

[37] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitras, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *Proc. 28th USENIX Conf. Secur. Symp.*, Berkeley, CA, USA: USENIX Association, Jun. 2019, pp. 497–514.

[38] O. Savry, M. El-Majihi, and T. Hiscock, "Confidaent: Control FLow protection with instruction and data authenticated encryption," in *Proc. 23rd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2020, pp. 246–253.

[39] M. Werner, T. Unterluggauer, D. Schaffenrath, and S. Mangard, "Sponge-based control-flow protection for IoT devices," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Apr. 2018, pp. 214–226.

[40] B. Kaliski, *Cryptographic Message Syntax Version 1.5*, document RFC2315: PKCS #7, 1998.

[41] R. Cantoro, N. I. Deligiannis, M. S. Reorda, M. Traiola, and E. Valea, "Evaluating the code encryption effects on memory fault resilience," in *Proc. IEEE Latin-Amer. Test Symp. (LATS)*, Mar. 2020, pp. 1–6.

[42] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural Comput. Appl.*, vol. 32, no. 4, pp. 1109–1139, Feb. 2020.

[43] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322–17341, 2017.

[44] W. Sung, S. Shin, and K. Hwang, "Resiliency of deep neural networks under quantization," Jan. 2016, *arXiv:1511.06488*.

[45] G. D. Jenssen, T. Moen, and S. O. Johnsen, "Accidents with automated vehicles—Do self-driving cars need a better sense of self," in *Proc. 26th ITS World Congr.*, 2019, pp. 1–10.

[46] *754-2019—IEEE Standard for Floating-Point Arithmetic*, Standard IEEE Std 754-2008, 2019.

[47] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, "A reliability analysis of a deep neural network," in *Proc. IEEE Latin Amer. Test Symp. (LATS)*, Mar. 2019, pp. 1–6.

[48] 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 *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2017, pp. 1–12.

[49] L. Van Winkle. *C Neural Network Library: Genann*. Accessed: Nov. 5, 2020. [Online]. Available: https://codeplea.com/genann

[50] N. C. Gokul. *DarkNet Classifier LeNet MNIST*. Accessed: Dec. 21, 2020. [Online]. Available: https://github.com/ashitani/darknet_mnist

[51] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.
[52] Y. LeCun. *LeNet-5*. Accessed: Nov. 6, 2020. [Online]. Available: http://yann.lecun.com/exdb/lenet/
[53] Y. LeCun, C. Cortes, and C. J. Burges. *The MNIST Database*. Accessed: Nov. 6, 2020. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[54] J. Redmon. *Darknet: Open Source Neural Networks in C*. Accessed: Nov. 6, 2020. [Online]. Available: http://pjreddie.com/darknet/
[55] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Apr. 2009, pp. 502–506.
[56] A. Ruospo, E. Sanchez, M. Traiola, I. O'Connor, and A. Bosio, "Investigating data representation for efficient and reliable convolutional neural networks," *Microprocessors Microsyst.*, vol. 86, Oct. 2021, Art. no. 104318.
[57] A. Ruospo and E. Sanchez, "On the reliability assessment of artificial neural networks running on AI-oriented MPSoCs," *Appl. Sci.*, vol. 11, no. 14, p. 6455, Jul. 2021.

NIKOLAOS IOANNIS DELIGIANNIS (Member, IEEE) received the M.Sc. degree in computer science and engineering from the Department of Computer Science and Engineering, University of Ioannina, Greece, in 2019. He is currently pursuing the Ph.D. degree with the Department of Control and Computer Engineering, Politecnico di Torino. He was a Research Assistant with the Department of Control and Computer Engineering, Politecnico di Torino. His research interests include testing of processors using formal methods and fault tolerance.

RICCARDO CANTORO (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer engineering from the Politecnico di Torino, Italy, in 2013 and 2017, respectively. He is currently a Researcher with the Department of Computer Engineering, Politecnico di Torino. His research interests include software-based functional testing of SoCs and memories, and machine learning applied to test and diagnosis.

MATTEO SONZA REORDA (Fellow, IEEE) received the M.Sc. degree in electronics and the Ph.D. degree in computer engineering from the Politecnico di Torino, Italy, in 1986 and 1990, respectively. He is currently a Full Professor with the Department of Control and Computer Engineering, Politecnico di Torino. He published more than 400 articles in the area of test and fault tolerant design of reliable circuits and systems, receiving several best paper awards at major international conferences. He is involved in numerous research projects with companies and other research centers worldwide.

MARCELLO TRAIOLA (Member, IEEE) received the M.Sc. degree *(cum laude)* in computer engineering from the University of Naples Federico II, Italy, in 2016, and the Ph.D. degree in computer engineering from the University of Montpellier, France, in 2019. From February 2020 to September 2021, he was a Postdoctoral Researcher at the Lyon Institute of Nanotechnology, École Centrale de Lyon, France. He is currently an INRIA Postdoctoral Researcher with the TARAN Research Group, IRISA Research Institute, Rennes, France. His main research interests include emerging computing paradigms with special interest in design, test, and reliability.

EMANUELE VALEA (Member, IEEE) received the M.Sc. degree in electronic engineering from the Politecnico di Torino, Italy, in 2016, and the Ph.D. degree in microelectronics from the University of Montpellier, France, in 2020. He is currently a Research Engineer at CEA-List, Grenoble, France. His research interests include hardware security and trust, cryptographic primitives for microelectronics and security-related aspects of VLSI testing and reliability.