Emulating the Effects of Radiation-Induced Soft-Errors for the Reliability Assessment of Neural Networks

Lucas Matana Luza*, *Student Member, IEEE,* Annachiara Ruospo*, *Student Member, IEEE,*
Daniel Söderström, *Student Member, IEEE,* Carlo Cazzaniga, Maria Kastriotou,
Ernesto Sanchez, *Senior Member, IEEE,* Alberto Bosio, *Member, IEEE,*
and Luigi Dilillo, *Member, IEEE*

Abstract

Convolutional Neural Networks (CNNs) are currently one of the most widely used predictive models in machine learning. Recent studies have demonstrated that hardware faults induced by radiation fields, including cosmic rays, may significantly impact the CNN inference leading to wrong predictions. Therefore, ensuring the reliability of CNNs is crucial, especially for safety-critical systems. In the literature, several works propose reliability assessments of CNNs mainly based on statistically injected faults. This work presents a software emulator capable of injecting real faults retrieved from radiation tests. Specifically, from the device characterisation of a DRAM memory, we extracted event rates and fault models. The software emulator can reproduce their incidence and access their effect on CNN applications with a reliability assessment precision close to the physical one. Radiation-based physical injections and emulator-based injections are performed on three CNNs (LeNet-5) exploiting different data representations. Their outcomes are compared, and the software results evidence that the emulator is able to reproduce the faulty behaviours observed during the radiation tests for the targeted CNNs. This approach leads to a more concise use of radiation experiments since the extracted fault models can be reused to explore different scenarios (e.g., impact on a different application).

Index Terms

Neural nets, Reliability, Approximate methods, Fault injection, Radiation effects

1 Introduction

Deep Learning and, in particular, Convolutional Neural Networks (CNNs) are currently one of the most intensively and widely used predictive models in the field of machine learning [1]. Nowadays, there is an incentive for pushing CNNs from the cloud to the edge devices and, in particular, for real-time safety-critical systems, e.g., in autonomous driving. Several recent studies have demonstrated that hardware faults induced by external perturbations (i.e., in a harsh environment) can significantly impact the inference leading to CNN prediction failures [2], [3]. Therefore, ensuring the reliability of CNNs is crucial, especially when deployed in safety- and mission-critical applications, such as robotics, aeronautics, smart healthcare, and autonomous driving.

Reliability assessment is always a costly phase. It is usually carried out by artificial Fault Injections (FIs) into the application under test. Depending on how faults are injected, we can build a simple taxonomy of FI techniques [4]. **Physical-based** FIs expose the system implementation to the same external conditions with respect to the in-field application, and therefore they guarantee a precise reliability assessment. However, they are also expensive in terms of hardware resources and exposure time. Moreover, they generally suffer from low controllability and observability, making it difficult to analyze the results obtained. On the other hand, **software-based** FIs modify the behaviour of the software to simulate hardware fault occurrences. The cost is lower compared to physical-based FIs with a higher degree of controllability and observability. However, the precision assessment of a software-based FI strictly depends on the adopted fault models. Finally, **model-based** FIs can be considered a kind of compromise between costs and precision. They work on a model of the application, i.e., usually, a hardware description language (HDL) model, where faults are injected during the model simulation or emulation. Therefore, injected faults are more close to a physical-based FI, but the simulation/emulation time can become quickly unpractical, even if high abstraction level models are used (e.g., Register-Transfer Level (RTL) model versus transistor model).

- *L. Matana Luza and L. Dilillo are with the LIRMM, Univ. Montpellier, CNRS, Montpellier, France. E-mail: {lucas.matana-luza, dilillo}@lirmm.fr*
- *A. Ruospo and E. Sanchez are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, Italy. E-mail: annachiara.ruospo@polito.it*
- *D. Söderström is with the Department of Physics, University of Jyväskylä, Jyväskylä, Finland.*
- *C. Cazzaniga and M. Kastriotou are with ISIS Neutron and muon source, Science and Technology Facilities Council, Didcot, England.*
- *A. Bosio is with Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, 69130 Ecully, France.*

$*$ *The authors contributed equally to the work.*

In this paper, an effective emulation platform for FIs is proposed. The injected faults are based on real fault models that are defined based on the outcomes of actual radiation test campaigns that have targeted an extensive characterisation of the Device Under Test (DUT). Since memories are the highest contributor of soft errors in systems [5], [6], [7], in this study, we focus our attention on this device, which, for this purpose, has been exposed to an atmospheric-like neutron beam. The DUT is a commercial DRAM memory that, in the system, has the role of storing the CNN parameters. As already pointed out, radiation-based FIs lead to a precise reliability assessment (i.e., the DUT is exposed to a realistic harsh environment) at a high cost in terms of hardware resources and exposure time. Therefore, we propose to leverage on a physical characterisation of the DUT through radiation experiments to extract the event rates and fault models. Thanks to this characterisation, we designed and implemented a software emulator to reproduce the incidence of such real faults and to evaluate their effect on CNN inferences. In particular, we target the use of CNNs for edge computing and, therefore, for low power embedded devices with limited resources. In our scenario, the CNN is implemented as a bare-metal application executed on a low power CPU. The ultimate goal is to have the flexibility of a software-based fault injector with a reliability assessment precision close to the physical one, thanks to the characterisation made on the actual device in realistic environmental conditions (through an accelerated neutron beam).

The rest of the paper is organised as follows. Section 2 presents related works that performed CNN reliability assessments. Section 3 describes our proposed approach. Subsequently, Section 4 presents the outcomes of the software-based fault injector. Finally, the conclusions and future directions are drawn in Section 5.

2 Related Works

In the literature, several works have been proposed in the last years for addressing the reliability assessment of CNNs. Concerning physical-based FI, we can cite [2], where the reliability dependence on three different Graphics Processing Unit (GPU) architectures (Kepler, Maxwell, and Pascal) was evaluated executing the Darknet Neural Network [8] when exposed to atmospheric-like neutrons. In [9], the authors analyse the reliability of a 54-layer Deep Neural Network injecting faults in the network weights and input data using an accelerated neutron beam for studying transient errors and FI tests to simulate permanent faults. The inferences target floating-point values, and the results show that object detection networks tend to generate wrong results when exposed to hardware faults. Among software-based FI, [10] presents an analysis of the error propagation through different abstraction layers. The authors used an open-source framework, Tiny-CNN [11], to inject faults in specific layers. Further examples of software-based FI techniques for CNNs are provided in [12], [13], [14]. A model-based FI approach is shown in [15], where the authors explore the resilience of an RTL model of a neural network accelerator. The work focuses on a High-Level Synthesis approach to characterise its vulnerability, injecting permanent and transient faults in various components in the RTL design.

A further framework developed to study the behaviour of hardware errors in deep learning accelerators is FIdelity [16]. It is able to model a class of hardware errors (transient) in software with high accuracy only leveraging on high-level design information obtained from architectural descriptions. Based on that, in FIdelity, it is possible to inject bit-flips in weights, input images and output neurons. In this context, it is also worth mentioning Ares [17], a framework for quantifying the resilience of DNNs. Ares targets permanent faults occurring in memory and injects errors at the application level: in weights, activators, and hidden states through bit-flips. Also, in [3], the authors applied several FIs on the configuration memory of the FPGA in order to understand the impact on the reliability of CNNs implemented on this type of device. Very recently, Xu *et al.* [18] proposed a FI framework running on a Xilinx ARM-FPGA platform to investigate the system exceptions that may occur due to the presence of permanent faults.

Our work proposes to leverage on the outcomes of precise physical characterisation tests to feed a software emulation platform for running fault injections campaigns with a reliability assessment precision close to the physical one. Radiation experiments on a CNN application with three different data representations are carried out to confirm the emulation platform consistency. Specifically, this paper presents two types of radiation experiments: one for the device characterisation to extract the real fault models and the other to verify the correctness of the emulator outcomes. The proposed emulation platform can be utilised for more in-depth analysis with different scenarios and applications. Finally, this approach leads to a concise use of radiation experiments, enhancing time and cost efficiency.

3 Proposed Approach

We strongly believe that the key point to achieve a meaningful reliability assessment relies on the accuracy of the injected fault models and the possibility to emulate their behaviour. For this reason, this work provides a comprehensive methodology for reproducing the incidence of real faults on a CNN application by relying on a software emulator. The flowchart of the proposed approach is shown in Figure 1. Starting from the characterisation of the target device under an external source of errors (*radiation experiment* #1) we extract a real set of fault models and event rates, which are used to configure and feed a software emulator. This software-based framework, described in Section 3.2, is capable of injecting the fault models extracted from the DUT characterisation on the parameters (i.e., weights and biases) of a CNN application. Therefore, in order to establish the effectiveness of the methodology, in parallel, radiation FIs (*radiation experiment* #2) are also carried out targeting the DUT storing the CNN parameters (Section 3.1.2). Finally, the outcomes from the emulator are analysed, and a correlation between the physical-based FI and software-based FI is provided in Section 4.

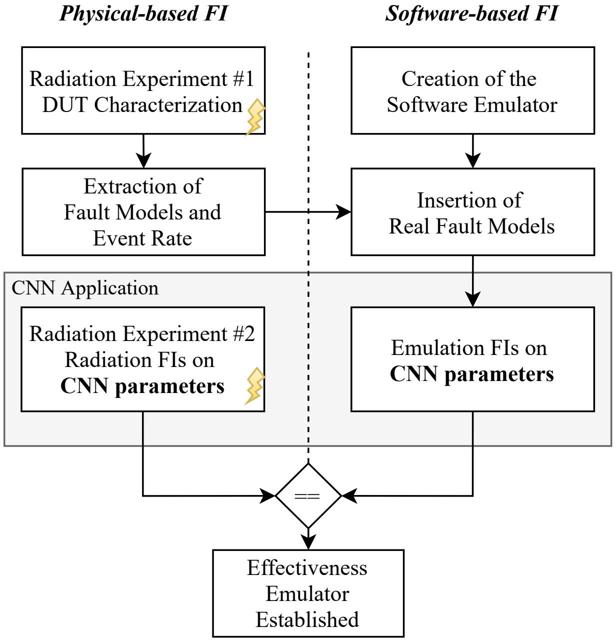
Physical-based FI
Software-based FI
Radiation Experiment #1 DUT Characterization
Creation of the Software Emulator
Extraction of Fault Models and Event Rate
Insertion of Real Fault Models
CNN Application
Radiation Experiment #2 Radiation FIs on CNN parameters
Emulation FIs on CNN parameters
==
Effectiveness Emulator Established

Fig. 1. Flow diagram of the proposed approach.

The noteworthy difference between the above-mentioned existing FI frameworks and the proposed one is the injection methodology and the closeness to physical injection. While most of the approaches that can be found in the literature inject random errors in a statistical way, our emulator injects faults based on realistic fault occurrences (for SBUs and stuck-at bits) or more real fault models (such as block errors) derived from radiation test campaigns. Moreover, the occurrence rate of the different fault models also reflects the actual experimental data (error cross section).

In the following, first, all the radiation experiments and related fault models are detailed in Section 3.1, then the architecture of the emulator is presented in Section 3.2.

3.1 Radiation Experiments

The radiation test campaigns [19], [20] were carried out at the Rutherford Appleton Laboratories, UK, where an atmospheric-like neutron spectrum is delivered in the ChipIr beamline at the second target station of the ISIS neutron source. The neutron flux in the beamline is approximately 10^9 times larger than the atmospheric neutron flux. The ChipIr facility provides a neutron flux of about 5×10^6 n/cm^2/s for energies above 10 MeV [21].

First, in Section 3.1.1 we provide an overview of the fault models that have been extracted from the DUT characterisation procedure detailed in [22]. Specifically, a memory device was characterised under an atmospheric-like neutron beam, and three different fault types were identified. Next, in Section 3.1.2, we present radiation tests on CNN applications and introduce a different test scenario, which is the focus of this work, where the DUT is storing only the parameters of a CNN (i.e., the weights and biases). With the presented scenario characteristics and correlating the results obtained by the DUT characterisation, we estimate the number of occurrences of the three different fault models during the execution of CNNs (Section 3.1.3). These outcomes are the basis for developing an emulator in which the real fault models identified under radiation are injected in CNN applications, leveraging software-based FI and allowing an in-depth analysis.

3.1.1 DUT Characterization

The DUT is the S27KS0642GABHI020, a 64 Mib HyperRAM™ self-refresh DRAM manufactured by Cypress Semiconductor. The evaluation of the device is based on the execution of two different memory test modes: static and dynamic. The static test mode consists of writing the memory cells with a known data pattern (e.g., solid ‘0’, solid ‘1’, and checkerboard patterns). The memory is then exposed to radiation to reach a defined particle fluence, and finally, a readback operation is performed to collect radiation-induced faults [23], [24], [25]. Withal, the application of a dynamic test mode aims at identifying functional faults during the emulation of real applications that constantly access the memory with write and read operations. The dynamic mode was executed using four different memory test algorithms: March C-, Dynamic Stress, Dynamic Classic, and mMats+ [24], [26]. The DUT was exposed to a cumulative neutron fluence of about 8×10^{11} n/cm^2, which, by considering the ChipIr average neutron flux, means approximately 45 h of experiments.

Three different types of faults were observed during the test executions. The first fault type consists of SBUs, appearing as a ‘0’ to ‘1’, and ‘1’ to ‘0’ transition. A write operation was sufficient to erase these SBUs, and they just occur once in the memory array. The second one recalls the stuck-at bit fault, which appears as permanent or temporary. The stuck-at bit fault

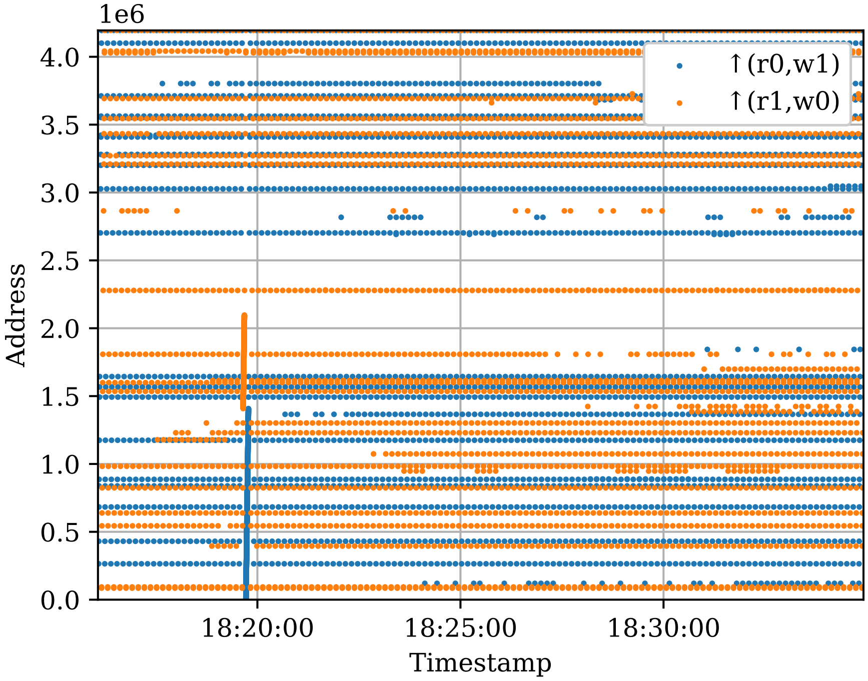

Fig. 2. Errors during a mMats+ test run. The dots represent a faulty word detected during the different operations of the algorithm [22].

results in memory cells that had their retention time affected by a particle interaction, resulting in a cell that always returns the same value. In DRAMs, both SBUs and stuck-at bits are suggested to have the same mechanism: degradation of the retention time capabilities of the cells, as presented in [27]. The third type of fault consists of block errors that cause a high concern for the application due to their extension. They appear in different shapes and affect different regions of the memory. This kind of fault was identified in four different patterns, with intermittent word errors in vertical and horizontal sequential addresses, affecting up to 2,048 addresses with the vertical pattern and 1,024 addresses with the horizontal pattern. These block errors are likely generated by a temporary malfunction of the sense amplifier or register that serves a column of the affected memory addresses. A more detailed view of this behaviour is described in [23].

Figure 2 represents the impact of the three fault models during the memory usage. It presents in the x-axis the timestamp of the operations realised during the execution of the mMats+ algorithm, in which the dots represent a faulty word detected during an element of the test. The y-axis presents the address of the faulty word in the memory, and each word is composed of 16 bits. It is possible to identify several stuck-at bits appearing as temporary and permanent during the test (horizontal sequences of dots on the graph), and a block error (appearing as two vertical sequences of dots) affecting 2,048 addresses.

To evaluate the memory sensitivity to the presented faults, the fault types were divided into SBUs, stuck-at bits, and block errors. The estimated event cross section (σ) is defined in two different ways. First, since the stuck-at bits and the SBUs are cell-related, the cross section takes into account the size of the memory, being the equation as:

$$\sigma_{\text{bit}} = \frac{N}{F \times M} \tag{1}$$

where N is the number of events, F is the cumulative fluence in particles/cm^2, and M is the number of bits [28]. When it comes to the block errors evaluation, since this fault is related to the control logic of the device, we may define the cross section as:

$$\sigma_{\text{device}} = \frac{N}{F} \tag{2}$$

where the memory size is removed from the equation, and the cross section is device-based. Moreover, the Soft-Error Rate (SER) is defined as:

$$SER_{FIT/Mb} = \sigma_{bit} \times (1,024 \times 1,024) \times 10^9 \times \phi \tag{3}$$

for the SBU and stuck-at bit, where $1,024 \times 1,024$ (bits) is the Mb coefficient, 10^9 is the Failures in Time (FIT) definition, and ϕ (13 particles/cm^2/h) is the neutron energies' ($>$ 10 MeV) flux at New York (sea level) outdoors for a mean solar activity defined in JEDEC JESD89A [29], [30]. Being slightly modified in the evaluation of the block errors SER, where the Mb coefficient is removed from the equation, which becomes:

$$SER_{FIT} = \sigma_{device} \times 10^9 \times \phi \tag{4}$$

Table 1 presents the estimated cross sections and SER for each fault type. The reader should note that the HyperRAM is not protected with ECC (Error Correction Code). Indeed, in embedded applications, its implementation may come at the cost of

TABLE 1
Estimated cross section with 95% confidence intervals using a fluence uncertainty of 10%, and SER for the fault types identified in this study. The values were calculated using the Eq. 1–4 [22].

Fault type	σ	σ Lower limit	σ Upper limit	SER
SBU	2.86×10^{-17} cm^2/bit	2.53×10^{-17} cm^2/bit	3.19×10^{-17} cm^2/bit	3.9×10^{-1} FIT/Mbit
Stuck-at bit	1.48×10^{-17} cm^2/bit	1.30×10^{-17} cm^2/bit	1.66×10^{-17} cm^2/bit	2.0×10^{-1} FIT/Mbit
Block error	4.48×10^{-11} cm^2/device	3.08×10^{-11} cm^2/device	6.23×10^{-11} cm^2/device	5.8×10^{-1} FIT

extra circuitry with power/performance overheads. For this reason, all occurrences of SBUs and stuck-at bits are considered without any filtering. Since CNNs have inherent resiliency to faults [31], the use of memory without ECC is justified. On the other hand, block errors may be catastrophic for a safety-critical application since they generate more than one bit-flip per word. Thus, in any case, the introduction of ECC would not be capable of protecting the system against this kind of fault. It is important to underline that the presented DUT characterization results can be exploited to set up a fault injector to evaluate the behaviour of any kind of application running on a computing system equipped with the chosen DRAM. Furthermore, the presented experimental methodology itself can be clearly applied to any other memory device.

3.1.2 Radiation Tests on the CNN Application

This section presents the radiation tests performed on a CNN application. These tests target the evaluation of the radiation-induced impact on the application and also are the base to verify the correctness of the emulator outcomes. To establish an actual case study, the analysis of the radiation-induced impact on a CNN application is done using three different neural networks with different data types based on the same LeNet-5 [32] architecture. Although proposed in 1998, LeNet-5 is currently being used as a valid and popular benchmark to introduce advances in research and present new ideas. And beyond this, the advantage of its adoption is mainly due to the size meeting the constraints of the embedded system environment with limited storage capacity. For the same reason, the simulation time required to run inferences during radiation tests are contained, with associated costs. Targeting a more complex embedded system, where the resources constraints are more relaxed, the use of deeper CNNs and larger datasets should also be considered.

3.1.2.1 CNNs implementation: The CNN architecture consists of three Convolutional Layers (CONV) followed by two Fully Connected (FC) layers and has a total of 61,470 parameters, of which 50% are in the convolutional layers. With respect to the original LeNet-5 structure, in our own models we removed the last SoftMax layer in order to bind the last FC layer to the classification output. We trained on the MNIST handwritten digit dataset by using 32 x 32-pixel cropped pictures. The training set contained 48,000 images, with an additional 12,000 for the validation set and 10,000 for the testing set. The learning rate started at 0.05, with the decay of 5×10^{-4} every $375(*128)$ iterations, and momentum was set to 0.9. The training was done by using the open-source framework N2D2 [33]. The LeNet-5 model description that we used is available in the framework itself. We define as “accuracy” of the CNN the capability to correctly classify the input picture. The accuracy is computed by using the top-1 score [32]. The achieved accuracy over the 10,000 testing images is 99%.

After the training, we exploited the N2D2 framework to export the trained network as C code using three different data representations:

Float 32 weights are represented by 32-bit floating-point real numbers;
Int 16 weights are quantized as 16-bit integers;
Int 8 weights are quantized as 8-bit integers;

For the last two data types, the quantization is done after training through the following steps.

1) all weights are rescaled in the range $[-1.0, 1.0]$ and activations at each layer are rescaled in the range $[-1.0, 1.0]$ for signed outputs and $[0.0, 1.0]$ for unsigned outputs;
2) inputs, weights, biases and activations are quantized to the desired n_{bits} by converting $[-1.0, 1.0]$ and $[0.0, 1.0]$ to $[-2^{n_{bits}-1} - 1$,$2^{n_{bits}-1} - 1]$ and [0, $2^{n_{bits}-1} - 1]$

Details are given in the N2D2 documentation [33].

The C code was ported to a Xilinx Zynq-7000 based system. This device is a System-on-Chip (SoC), which provides an ARM Cortex™A9 processor attached to a 28 nm Artix®7 FPGA. The CNN application was ported to this embedded system with the use of three external memories: two units of the MT41K128M16JT-125, which is a 2 Gb SDRAM DDR3L from Micron Technologies, and the DUT (HyperRAM) described in Section 3.1.1. Furthermore, to increase the reliability of the system, the SoC configuration memory (CRAM) was monitored by the commercial Xilinx scrubber, the Soft Error Mitigation (SEM) core, which reports detected SBUs, and, when possible, corrects them [34]. Figure 3 depicts the top-level diagram of the system.

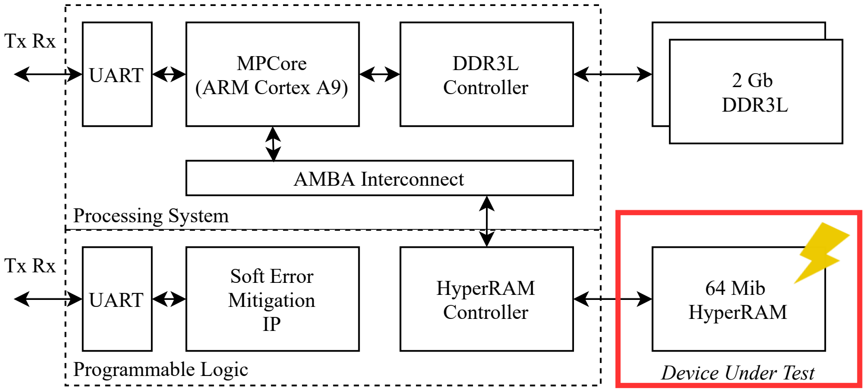

Fig. 3. System top-level diagram highlighting the DUT.

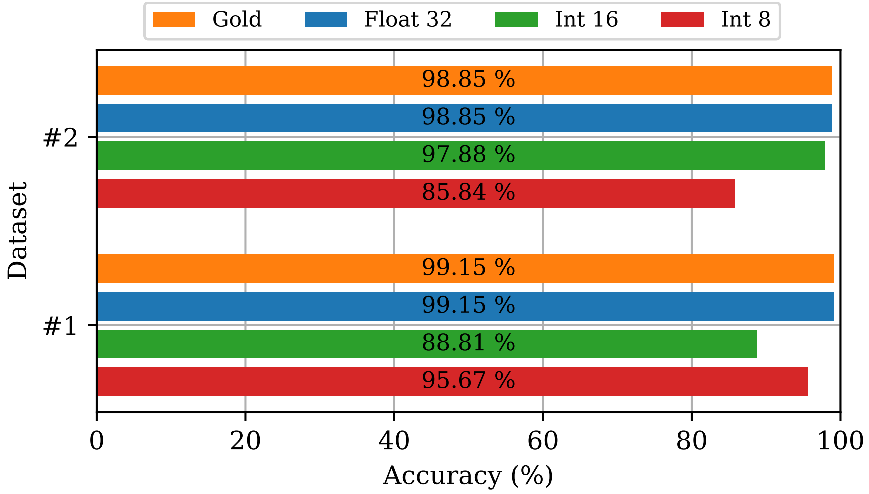

Fig. 4. Accuracy obtained from the results of the previous work.

3.1.2.2 Test procedure: The testing set was processed by the three versions of the network. Depending on the data representation, the accuracy of the CNN may present a slight degradation compared to the precise CNN (i.e., the one using 32-bit floating-point representation). The obtained accuracy based on the testing set (10,000 images) was 99.07% for *Float 32* and *Int 16*, and 99.05% for the *Int 8* version. It is interesting to point out that the accuracy degradation is really minor and affects only the *Int 8* data representation, which was expected because of the intrinsic error resilience of the CNN. On the other hand, the memory footprint presents a significant reduction thanks to the approximation in the data representation.

Since the memory layout for an application is divided in sections [35], which generally are:

text	executable instructions,
data	constants and statically allocated variables,
heap	dynamically allocated variables,
stack	store parameters for function calls, return addresses, and local variables.

The memories sections of the CNN code were split into two memory devices. In our previous work [22], the HyperRAM stored all data related contents, which basically include the *data* sections, and the *heap*, allocating 56.69% for *Float 32*, 53.70% for *Int 16*, and 52.17% for *Int 8*, of the HyperRAM capacity. This scenario was tested with two different portions of 2,000 images randomly chosen from the training set. The outputs during the irradiation were then compared with a golden result in order to identify faulty executions. For the three CNN versions, the golden accuracy for both datasets was the same, 99.15% and 98.85%, respectively.

The total accuracy obtained in the previous work is presented in Figure 4. This graph considers the total images per data representation and targets the dataset portion used during the runs.

The accuracy degradation on the previous work resulted from errors occurring in two different ways, affecting the software execution (when the output frame is corrupted) and affecting the CNN inference (when the network is not able to return the expected result). It is important to underline that after each execution of the CNN, the whole system was reprogrammed (FPGA bitstream reloaded, code memory sections rewritten, processor reset), preventing the Soft Error accumulation. Thus, this faulty behaviour is not due to stuck-at bits accumulated in the memory along with the irradiation, which was not able to impact the network since a gradual degradation of inference capability was not observed. On the other hand, considering the calculated error cross section, which is relatively low, and taking into account that the memory usage does not correspond to 100% of the memory storage capacity, we may correlate the faulty runs mainly to the effects of block errors in the memory. However, a more in-depth analysis is required to better understand the correlation between the fault occurrence in the memory with a faulty execution of the application. In the scenario provided in [22], only the *text* and *stack* sections were not exposed to the neutron beam, and the other variables and CNN parameters were a source of errors. As a result, the analysis capability was limited.

Then, in order to restrict the source of errors and enable a higher capability analysis, we define a scenario where only the *rodata (read-only data segment)* is allocated in the HyperRAM. This section contains static constant data and is mainly composed of CNN layers' weights and biases. Moreover, it uses approximately 505 kB in the *Float 32* CNN, 273 kB in *Int 16* CNN, and 154 kB in *Int 8* CNN. The test setup is described in Figure 3. In this case, the CNN parameters (weights and biases) are stored in the DUT according to the compiled code with GCC, in which we have the following relation: *Int 8*, 1 weight/bias, 1 byte; *Int 16*, 1 weight/bias, 2 bytes, and in *Float 32*, 1 weight/bias, 4 bytes. Furthermore, during the radiation test campaigns, the boards were mounted in the setup in a configuration that exposed only the DUTs to the beam, with more than one DUT at the same time.

In addition, to have a more in-depth view of the memory behaviour, at the end of each run, a readback operation in the *rodata* contents is performed, enabling the identification of SBUs, stuck-at bits, and block errors.

Henceforth, "run" is defined as the inference of a set of images (e.g., 2,000 or 1,000 images), starting from image '0' up to the last image within the dataset or when the execution is stopped. This definition is done since some runs did not achieve the end of their execution once a functional interruption occurred, which did not affect the DUT (HyperRAM), but other parts of the computation system. In these cases, we consider for calculation only the processed images within a run.

During the first part of the test, the target was the inference of 2,000 images per run. However, mostly for the *Float 32* version, which demands more time to be completed, the computing system suffered a functional interruption, and the execution failed before its end, resulting in runs without the readback operation. The radiation experiments were performed under an atmospheric-like neutron beam, and the ionisation in silicon is not directly generated by the neutron interaction with the matter but through neutron-induced silicon recoils and/or nuclear reactions byproducts. The created free charges of electron-hole pairs generated from the neutron events might lead to single-event effects in the devices [36], [37]. For example, the scattering of a neutron by collision with an atomic nucleus will lead to a neutron scattering at an angle as well as the nucleus recoils [38]. In this case, some scattered particles may hit not only the DUT but also some other electronic components of the system that are located in the irradiation room. Then, considering that the execution time of the *Float 32* version is higher when compared to the other versions (*Int 8* and *Int 16*), and adding it to the time needed to complete the 2,000 inferences, the probability to have some particles hitting other parts of the setup electronics increase within a run with these characteristics. The dataset was then reduced to 1,000 images, to increase the possibility of having complete runs. Additionally, the data were filtered in order to consider only the part concerning the inferences before the controlling system failure. The data concerning each inference are independent, so all the data coming up to the last non-failing inference (at the controlling system level) can be considered good for the analysis. This choice was adopted in order to optimise the use of the available beam time by stopping the ongoing run after a fixed timeout in the communication between the controller part, and the acquiring system and then starting a new run. Furthermore, concerning the failure in the controlling part of the system, this was not explored since it was out of the scope of this study.

From the total number of runs, 57.96% were completed. For the three CNN versions, the golden accuracy for both datasets was the same, 99.15% (2,000) and 99.30% (1,000). During the tests, the number of inferences for each CNN version was similar. The number of inferences were 44,370 for *Float 32*; 57,680 for *Int 16*, and 52,595 for *Int 8*.

3.1.2.3 Test results: Figure 5 presents a summary of the readback operation performed in the memory addresses containing the CNN weights and biases. The graph presents the number of errors identified in these addresses after the execution of a run of the CNN. From the readback operation, a bitmap is generated and presented the same effects on the memory running the CNN and the ones from the DUT characterisation test. From the five-block identified errors, all of them appeared as a vertical line of errors. In Figure 5, they are represented as the five highest points (above 10^2 in y-axis), and they were identified only in the *Int 16* version. Most of the runs with *Float 32* version did not reach the readback operation, which may result in the lack of identification of this kind of effect. Also, for the *Int 8* version, the memory usage is low, and it is kindly possible that block errors occurred in the memory but in an address range that was not used by the application. An example of a block error is depicted in Figure 6, where the red lines are used to delimit the region. The grey zones represent the memory region used by the CNN application, and each bit identified with an error appears as a black pixel. A zoom-in is added to enhance the bitmap visibility.

The total accuracy for the proposed scenario is presented in Figure 7. This graph considers the total images per data representation and targets the dataset size used during the runs. The expected result is presented, showing degradation in the *Float 32* and *Int 16* versions. All the runs with the *Int 8* version returned the expected value. Additionally, from the total runs, 6.37% presented a difference in the final accuracy. In which, Table 2 summarises these faulty runs by presenting the first image with a faulty inference and the final accuracy obtained from this run.

From these faulty runs, one case, in particular, was able to increase the final accuracy by resulting in the correct inference of some digits when compared with a non-faulty (gold) run, where the inference results in a wrong prediction. This behaviour occurred for two images in the faulty run number 10 (according to Table 2), where the gold run returns the digit '3', being the expected the digit '5'. This run, where a block error was identified, resulted in the right value ('5') for both images, and the final

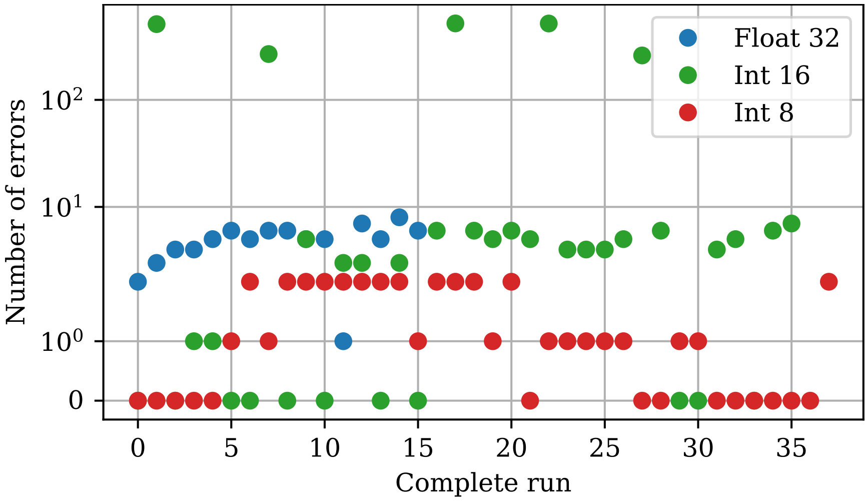

Fig. 5. The number of bytes with errors identified with the readback operation performed at the end of each run.

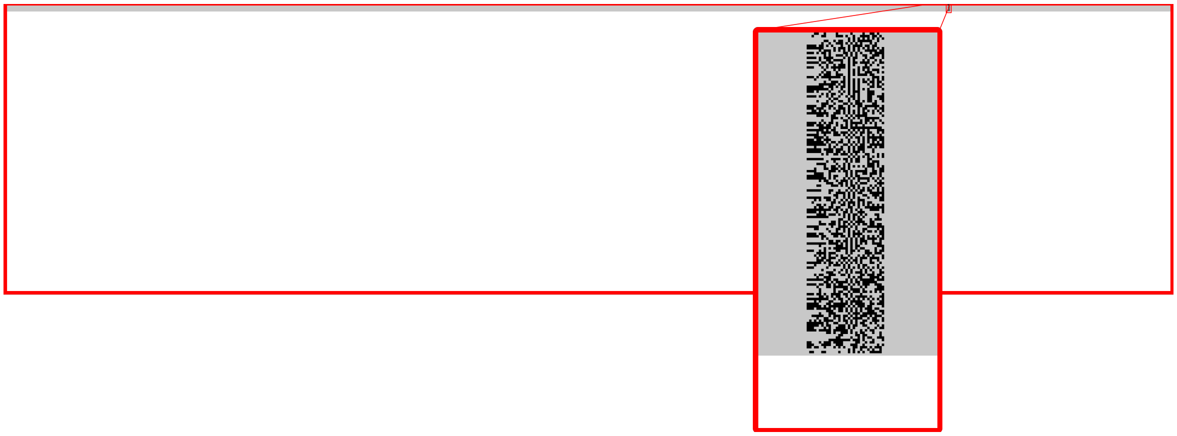

Fig. 6. Bitmap obtained from a readback operation after a run of a CNN *Int 16* version. Each pixel represents a bit; each bit that was identified with an error appears in black. The red lines are used to limit the region. The grey zone represents the memory usage. Zoom-in is added to increase the visibility of the block event.

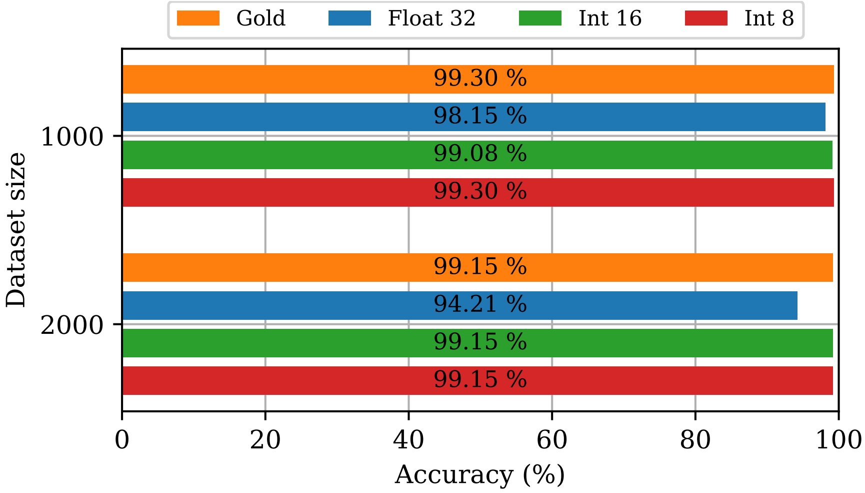

Fig. 7. Accuracy obtained from the results of the proposed test scenario.

accuracy increased from 99.3% to 99.5%. This case is really interesting and proves that CNN resilience assessment is quite different with respect to classical applications. The difference stems from the fact that CNNs are always an approximation of a function, in our case, a classifier. Thus, a fault can deviate the behaviour of the CNN not only from good to wrong prediction

TABLE 2
Summary of the runs that return a faulty accuracy based on the proposed scenario tests.

Faulty run	CNN version	Dataset size	First faulty inference [image no.]	Run accuracy [%]	Expected accuracy [%]
1	Int 16	2,000	1,345	99.10	99.15
2*	Float 32	2,000	1,384	84.36	99.09
3	Int 16	2,000	583	99.10	99.15
4*	Float 32	2,000	120	19.96	99.24
5*	Float 32	1,000	720	98.91	99.24
6*	Int 16	1,000	988	98.00	99.30
7	Int 16	1,000	102	99.20	99.30
8	Int 16	1,000	720	93.60	99.30
9	Float 32	1,000	673	70.40	99.30
10	Int 16	1,000	189	99.50	99.30

* Incomplete runs.

(the majority of the cases), but also from **wrong** to **good** prediction.

The second case is related to an execution based on the 32-bit floating-point version (faulty run number 5), where the output vector returned all the values as a floating-point exception (NaN, “Not a Number”), which resulted in an invalid floating-point operation. This behaviour invalidated the top-1 score, and the final output was defined as a digit ‘0’. Again, this case is really interesting because it cannot be considered as a usual silent data corruption (that is, by definition, non-detectable). Here, by analysing the CNN outputs, it is possible to detect the faulty behaviour and thus trigger an error signal/trap.

For the other faulty runs, targeting the complete ones, in four of them, a block error was identified during the readback operation. The other two presented a particular behaviour. First, run number 7, the output vector presented the corrected top-1 score, but the output application returned a faulty inference, giving a digit ‘0’ instead of ‘2’. This result may be correlated with a register or a variable allocated in the external memory (DDR3). Second, run number 9, after the first faulty inference, the output vectors returned the same wrong value for all subsequent inferences, having the digit ‘3’ as a result of the top-1 score. Also, it is interesting to highlight that the incomplete runs presented similar behaviour.

3.1.3 Execution Soft Error Rate Definition

To extract the expected number of events within a run, based on the SER (3), we define the Execution Soft Error Rate (E-SER) as:

$$E - SER_{\text{SBU | stuck-at}} = \sigma_{\text{bit}} \times M \times \bar{\phi} \times t \quad (5)$$

where σ_{bit} is the calculated cross section for SBUs or stuck-at bits, M is the memory size in bits used by the application (stored in the DUT), $\bar{\phi}$ is the average neutron flux during the test campaigns, and t is the application execution time in seconds. The same approach is used to define the E-SER related to block errors, in this case, since the block errors are device related, the rate is defined as:

$$E - SER_{\text{BE}} = \sigma_{\text{device}} \times \bar{\phi} \times t \quad (6)$$

From the E-SER definition, it is possible to calculate the expected event rate based on the presented radiation tests. The calculated values are presented in Table 3.

The results obtained in the previous work contrast with the ones obtained in this work. The previous work presented a higher E-SER. This difference is mainly due to the difference in memory usage, which is about 10x higher. Although, it is interesting to highlight that a direct comparison between the tests is inhibited by the difference in the implementation. In the previous work, all data related sections were stored in the DUT. However, in the scenario proposed in this work, the target is only the *rodata* section. The *rodata* is composed of static read-only contents, in contrast with the first implementation, which had also, e.g., the *heap* section, which is a dynamically allocated region in which read and write operations are performed. This difference can impact, e.g., the SBU analysis, since a write operation will recover the error, in contrast with a read-only section, where the SBUs will accumulate, and only a reprogramming in the memory contents can restore the SBU effect. The results obtained in this proposed scenario show that faults occurring in the CNN parameters (weights and biases) are more relevant when using a floating-point data representation. Also, an outcome from the previous work, where the most affected implementation was the *Int 8*, is that faults occurring in other data sections of the memory code had a larger effect on the resilience of the CNNs. A more in-depth analysis of the outcomes of the radiation tests proposed in this work is provided in the next sections.

3.2 Emulator

Three types of radiation-induced faults were extracted from the execution of static and dynamic test modes on the HyperRAM memory: SBUs, stuck-at bits, and block errors. Since the occurrence of such faults was independent of the running application (the characterization tests and the CNNs were running on separate DUTs in the radiation tests), a software

TABLE 3
Estimated event rate for the both test scenario. The results are calculated based on the definition of (5) and (6).

Test scenario	Dataset size	CNN version	E-SER [events/run]		
			SBUs	Stuck-at bits	Block errors
[22]	2,000 images	Float 32	11.57	5.99	0.47
		Int 16	7.97	4.12	0.34
		Int 8	6.05	3.12	0.27
This work	1,000 images	Float 32	0.84	0.43	0.32
		Int 16	0.24	0.12	0.17
		Int 8	0.08	0.04	0.10
This work	2,000 images	Float 32	1.53	0.79	0.59
		Int 16	0.48	0.25	0.34
		Int 8	0.17	0.08	0.21

emulator was designed to reproduce their incidence and to assess their effects on CNN applications. Particularly, it can be used to obtain more information that cannot be extracted from physical-based fault injections, such as the impact of faults on CNN internal structures (e.g., layers, channel or even kernel) or worst-case scenarios.

3.2.1 Architecture

The emulator was built with a multi-threading structure to mimic the process of the radiation campaigns and the neutron flux. A thread is a task linked to the main process but running independently from it, with its own stack pointer, registers, and thread-specific data. Although they exist as independent entities within a process, they can share the same resources, such as the same memory locations. Therefore, the adoption of multi-threading programming gave us the possibility to design an emulator as close to reality as possible. Its architecture is illustrated in Figure 8. Alongside the main process, it makes use of two threads: the inference (*thread1*) and the injector (*thread2*). They are created in the main process by calling specific functions belonging to the *pthread.h* library. Once created, they run independently to accomplish specific tasks. For instance, the only objective of the inference thread is to complete the inference of *N* images (also referred to as stimuli). On the contrary, the duty of the injector thread is to introduce faults in the memory locations. As stated, such resources are shared between the main process and the different threads. As an example, the network parameters (weights, biases, and images) are accessible by all threads. To avoid parallel read and write of shared data, the injector thread makes use of mutexes to achieve thread synchronization and, therefore, safe communications. In other words, when *thread2* has to introduce a fault in the memory (e.g., a stuck-at 0 in a weight), it takes the mutex, injects the fault, and then releases the mutex. At this juncture, while the mutex is taken, *thread1* is not allowed to read the shared memory, resulting in a pause of the inference process.

Moreover, to emulate a neutron flux and obtain realistic injections, *thread2* introduces faults only with a random frequency determined by the variable *randtime*. Before injecting, it sleeps for a random time (by exploiting the *sleep* function), and then the process is repeated for the next fault to inject. In this way, the amount of affected images is not fixed, neither the injection time. Clearly, to ensure that the injection occurs within the inference window, a maximum bound is given to the sleep function. We would like to stress the fact that the possible amount of injections cannot be enumerated. It is therefore crucial for the characterization phase developed in the previous section to guide the injector and consider the realistic cases. In this regard, before creating the threads, the emulator is configured to deal with a specific E-SER, detailed in Section 3.1.3, and a definite fault model. It can work with the three observed fault models: SBUs, stuck-at bits, and block errors. While for SBUs and stuck-at bits, the faulty location, as well as the injection time, are randomly selected, emulating the block errors requires a specific parser. As described before, e.g., if up to 2048 sequential memory addresses are affected by an error, a block error is identified in a specific region of the HyperRAM memory. However, to inject these block errors into the application parameters, it is first necessary to see whether their corresponding addresses match (the physical addresses returned from the radiation test campaigns with the virtual addresses of the application). For this purpose, a specific parser was created to map the physical addresses of the memory to the virtual addresses seen by the main process.

Faults in the HyperRAM memory are reported at the end of the radiation test campaign. These reports include for each corrupted memory address: a timestamp, an identifier related to the target experiment, a counter named dynamic cycle, the physical address affected by the fault, the corresponding error data, and the specific cycle of the dynamic test. Then, starting from those sets of reports and knowing the virtual addresses of the CNN application, the parser was created and included in the emulator architecture. The fault lists that are provided to the emulator have rows in the form:

$$< physical\_address >< bit\_position >< bit\_type >$$

Then, the emulator internally maps the addresses from physical to virtual and, if there is a match, the block errors are injected in the corresponding memory addresses of the application. The emulator can inject in both weights and biases of the CNN applications, the same portion of data that are allocated in the *rodata* memory section under radiation (Section 3.1.2).

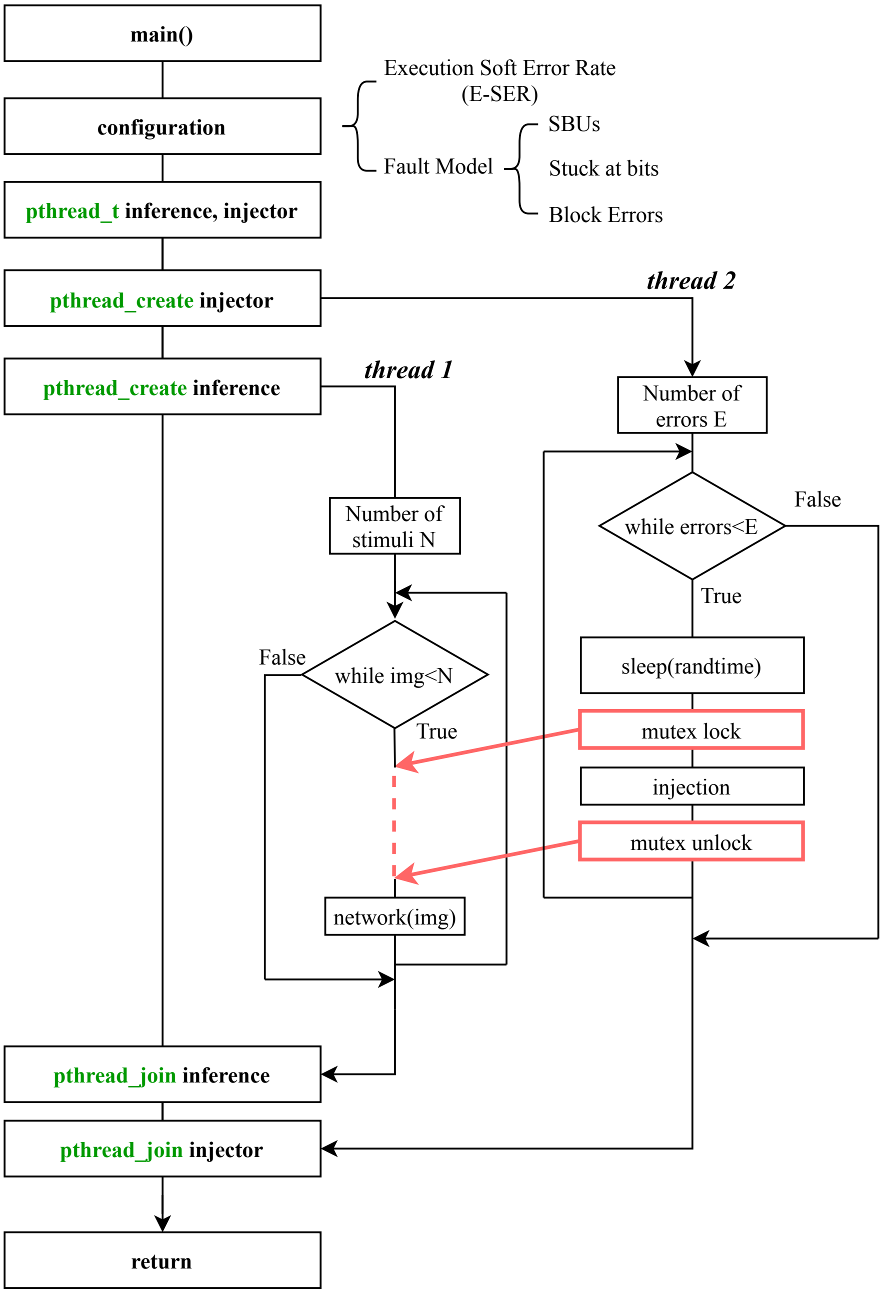

Fig. 8. Emulator block diagram.

4 Results Analysis

In this section, a correlation between the results obtained during the radiation experiments is done alongside the analysis of the outcomes from the emulator experiments. To prove the effectiveness of the proposed emulator, experiments were performed on three different CNN applications based on the LeNet-5 model; the same CNNs that were used for the radiation experiments described in Section 3.1.2. They exploit three different data representations for representing the neural network parameters: 32-bit floating-point (*Float 32*), 16-bit integer (*Int 16*), and 8-bit integer (*Int 8*). As specified in Section 3.1.2, they are written in C code and are trained with the open-source N2D2 framework [33]. Since the emulator can inject in both weights and biases parameters, we exactly reproduced the configuration of the proposed radiation test scenario, where the *.rodata* section of the memory hosting weights and biases was under radiation. Therefore, the same dataset of 1,000 images was chosen from MNIST for a single run. For every CNN application, the injection campaigns of the three fault models (SBUs, stuck-at bits, and block errors) were driven by the event rate parameter defined in Section 3.1.3 and calculated in Table 3.

The experiments with the software emulator were performed on a Linux server equipped with a dual Intel Xeon CPU E5-2680 v3 and 256 GB of RAM. To perform a golden run (i.e., the inferences of 1,000 images from MNIST), the software emulator took about 200 ms in all three CNNs. While, due to the usage of mutexes, it took a maximum of 16 s to inject the block errors (the most extended ones) during a run.

TABLE 4
Details of SBUs and stuck-at bits injection for the CNNs *Float 32*, *Int 16*, *Int 8* with an increasing E-SER: 1x, 25x, 50x, 75x, 100x

E-SER	**Total Runs**	**CNN Float 32**			**CNN Int 16**			**CNN Int 8**		
		SBUs/run	**Stuck-at bits**		**SBUs/run**	**Stuck-at bits**		**SBUs/run**	**Stuck-at bits**	
			Stuck-at/run	**Total**		**Stuck-at/run**	**Total**		**Stuck-at/run**	**Total**
1x	50	0.84	0.43	25	0.24	0.12	6	0.08	0.04	2
25x	50	21	11	550	6	3	150	2	1	50
50x	50	42	21	1,050	12	6	300	4	2	100
75x	50	63	32	1,600	18	9	450	6	3	150
100x	50	84	43	2,150	24	12	600	8	4	200

4.1 SBUs and Stuck-at Bits

The first set of experiments dealt with SBUs and stuck-at bits for all the targeted CNNs. For each of them, the emulator was configured to carry out five different experiments with the two fault models. First, the E-SER retrieved from the radiation test campaigns was used to assess the CNNs reliability. Then, it was tuned to study worst-case scenarios on the same CNN application. We aimed at evaluating the CNNs resilience when a growing number of SBUs or stuck-at bit affected the network weights and biases at a random time. Both the faulty weight/bias location and the bit position within it are randomly chosen. Then, SBUs are injected through bit-flips, while stuck-at bits introduce a permanent 0 or 1 in the specific bit position of a given weight or bias.

The figures for the injection campaigns are provided in Table 4. At the beginning (row *1x*), the emulator was set up with the nominal E-SERs for both SBUs and stuck-at bits (the one calculated in Table 3). Then, the E-SER was incremented by 25 times (*25x*), 50 times (*50x*), 75 times (*75x*), and 100 times (*100x*). Columns 3 and 4 from Table 4 report the number of SBUs and stuck-at injected during each run. Since stuck-at bits are accumulated, in Column 5 it is reported the total amount of faults that have been injected at the end of all runs. The same reasoning is applied to the other two CNNs. It is worth reminding that a run is defined as the inference of a set of 1,000 images from MNIST. A golden run (run without injected faults) over this set achieves the 99.30% accuracy for the three CNNs.

As for the nominal E-SER (*1x*), injecting a non-integer number of SBUs or stuck-at bits during a run was not feasible. Therefore, they have been approximated as follows:

- *CNN Float 32*: For SBUs, one single random fault was injected during a single run; for stuck-at bits, one single fault was injected every 2 runs.
- *CNN Int 16*: For SBUs, one single random fault was injected every 4 runs; for stuck-at bits, one single fault was injected every 8 runs.
- *CNN Int 8*: For SBUs, one single random fault was injected every 16 runs; for stuck-at bits, one single fault was injected every 32 runs.

The big difference was that, while for SBUs, the fault remained active only during a single run, the stuck-at bits were accumulated over the runs. For each of them, the experiment was repeated 50 times with SBUs or stuck-at affecting random fault locations (weights or biases), layers and bit positions. The same reasoning was applied for higher E-SER (*25x*, *50x*, *75x*, *100x*) by injecting SBUs or stuck-at bits according to the terms specified in Table 4.

Experimental results are shown in Figures 9 and 10, where the average accuracy value is depicted for the three CNNs under assessment, along with the minimum and the maximum value achieved during the 50 runs (small error bars). At first glance, it is apparent that the CNN application most affected by the occurrence of stuck-at bits and SBUs is the *Float 32*. Indeed, as the number of faults increases, the *Float 32* accuracy considerably decreases. This effect confirms the outcome of the radiation test campaigns, where the *Int 16* and the *Int 8* CNNs show greater resilience. Then, regarding the nominal E-SER (*1x*), data show that the final network accuracy for the three CNNs was not affected for stuck-at bits, staying at 99.30%. For SBUs affecting the *Float 32* CNN, the final accuracy was equal to 99.29%: only a single run among the fifty was faulty. This is in line with the results coming from the radiation test campaigns. For instance, regarding the *Float 32* CNN application under neutron radiation, despite the occurrence of SBUs or stuck-at bit faults (as highlighted in Figure 5), only 1 complete run over 17 was faulty. The remaining 16 were able to withstand such radiation-induced faults without compromising accuracy. Besides, with the nominal E-SER (*1x*), the two integer CNNs (*Int 16* and *Int 8*) keep the golden accuracy (99.30%) over the 50 runs, in line with the radiation test campaigns results. Indeed, as shown in Figure 5, despite some errors appear at the end of the inference process of *Int 16* and *Int 8* CNNs, in the latter case, they do not affect the final accuracy percentage at all. In the former, the accuracy degradation that we discuss in Table 2 is more related to the occurrence of block errors. Next, as the E-SER increases, the final network accuracy considerably decreases in the *Float 32* CNN. Contrarily, the increase of the E-SER slightly influences the final accuracy value in both *Int 16* and *Int 8*. Only with a E-SER *100x* we observe a small degradation (respectively 1% and 2%) when the *Int 16* and *Int 8* CNNs are affected by stuck-at bits.

Generally, the graphs point out that stuck-at bits are more critical than SBUs. Indeed, once stuck-at bits appear, they accumulate over 50 runs. A last observation in the *Float 32* CNN is related to the scenarios *75x* and *100x* for stuck-at bits. Although the quantity of injected faults was higher in the second case, the results suggest that the choice of the faulty bit positions (random in our experiments) plays a crucial role in the final accuracy assessment. Consistent with the results achieved in [10], [13], we found that faults affecting the exponent bits are the most critical, leading in many cases to NaN ("Not a Number") values. This specific outcome mirrors and perfectly reproduces the faulty behaviour that was observed during the radiation tests, as described in Section 3.1.2.

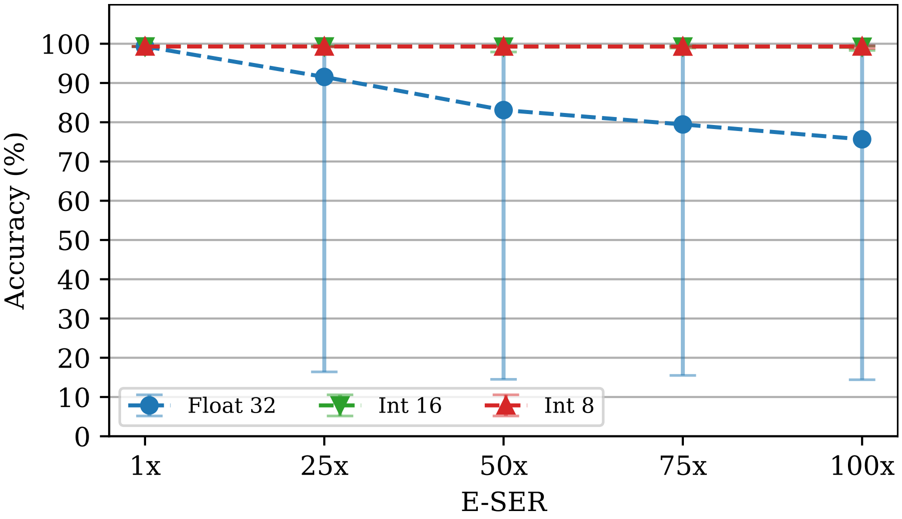

Fig. 9. Accuracy variation based on the increase of the SBUs E-SER for the three CNNs.

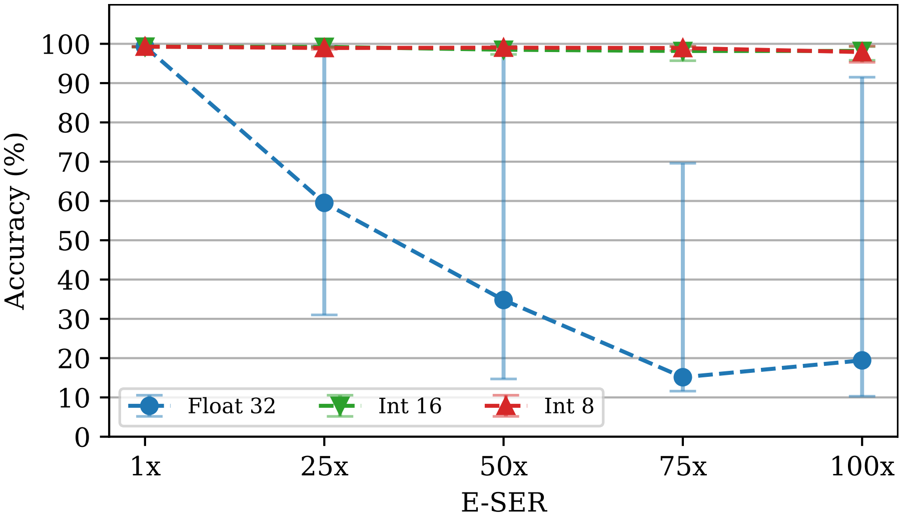

Fig. 10. Accuracy variation based on the increase of the stuck-at E-SER for the three CNNs.

4.2 Block Errors

From the application of static and dynamic characterization tests during the atmospheric-like neutron radiation (Section 3.1.1), a total of 37 block errors (BEs) were identified in the HyperRAM memory. The purpose of the following experiments is to inject these block errors into the three CNN application parameters (weights and biases), only if their physical and virtual addresses match. To this end, the parser described in Section 3.2.1 is used, and if there is a correspondence between addresses, the emulator injects a stuck-at bit in the specific bit position of the respective weight or bias. For the sake of clarity, the fault lists are not randomly generated (as for SBUs and stuck-at bits), but they derive from the radiation test campaigns. In Table 5, for each CNN application, we provide details on the number of BEs matching the addresses, the total faults that can be introduced into the application parameters, and, for the sake of completeness, the memory footprint of the given CNN application in the HyperRAM memory. The number of injected faults for each single BE is illustrated in Figure 11, looking at the y-axis on the left.

It is also worth noticing that their incidence can also vary from a few faults injected (e.g., BE1 and BE2 with only 8 faults in the *Int 8* CNN) up to many ones (e.g., BE29 with 972,960 in the *Float 32* CNN).

TABLE 5
Block Error Injection Details

CNN Application	Total Injected BEs	Total Injected Faults	Memory Footprint (kB) *(.rodata)*
Float 32	17	3,278,656	505.81
Int 16	11	1,714,007	273.17
Int 8	15	605,922	154.11

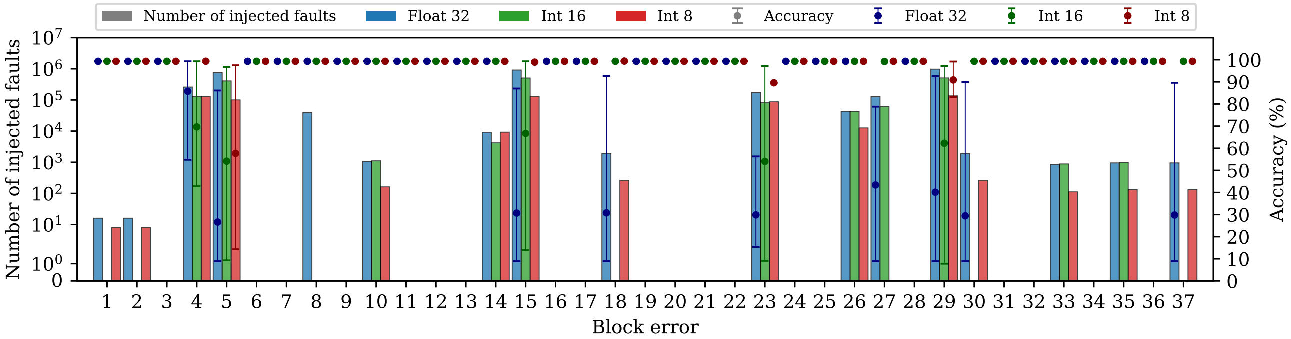

Fig. 11. Number of injected stuck-at faults for each single block error for each CNN application (y-axis on the left). Average accuracy value for each single block error affecting at a random time a run (y-axis on the right).

First, we calculated for each CNN the impact of each single BE occurring during a run at a random time. Dealing with random injection times, each block error was injected 50 times in 50 different random moments (during the run of 1,000 inferences). The y-axis on the right of Figure 11 shows the average accuracy value achieved at the end of the 50 runs. The error bars highlight the minimum and maximum accuracy values. Particularly worth of note is the minimum: it was computed at time zero, meaning that the BEs were injected before running the 1,000 inferences. In ten cases (seven in *Float 32* and three in *Int 16*) it was less than the 10%. The maximum of the error bars represents the accuracy obtained when the random time was close to the end of the 1,000 inferences and in some cases it was slightly higher than the golden. Considering a block error that affects the inferences, the impact on the final accuracy will clearly have a dependency on the moment that these faults are injected. Since the final accuracy is based on the 1,000 inferences result, the fault injection occurring more close to the beginning of the run will cause a more drastic impact on the final accuracy, and the opposite is also true. From further and more in-depth analyses, we made the following considerations for the three CNN applications. A set of BEs, once injected, led the CNNs to predict all subsequent images wrongly. Thus, they are referred to as critical for the given CNN application. In detail, the most criticals for each CNN are:

- *Float 32*: BE5, BE15, BE18, BE27, BE29, BE30, BE37
- *Int 16*: BE5, BE15, BE23, BE29
- *Int 8*: BE5

In the following, we analyze the reasons leading to this critical behaviour, and, generally, we discuss the influence, as well as the effects that block errors have on the targeted CNN applications. Concerning the *Float 32* CNN, the listed BEs inject stuck-at-1s into the exponent part of the floating-point values in one or multiple bits of the affected weights or biases. The corruption of such bits produced many floating-point exceptions (NaN) from which the network was not able to recover anymore. Interestingly, this reproduced a real scenario that occurred for the faulty run number 5 during the proposed radiation test campaign, detailed in Table 2. This behaviour (NaN exceptions) is observed for all the above critical BEs. The other two BEs leading to a degradation in the *Float 32* CNN accuracy but not included in the list of the most critical (i.e., BE4 and BE23) did not produce NaN values, and thus, the network was still able to yield some correct predictions, despite the BEs. They inject stuck-at-0 values. Finally, as shown in Figure 11, the remaining 28 BEs did not affect the final accuracy at all (staying at 99.30% despite the injected BE). The reasons are of two types: they do not inject any faults into the network parameters, or they inject stuck-at faults into the mantissa part of the floating-point values, which did not lead to a change in the final CNN prediction.

As for the *Int 16* CNN, BE15 and BE23 are of particular interest. In the addresses specified by their respective fault lists, rows of stuck-at-1 (BE15) or stuck-at-0 (BE23) are injected in all bits of the matching weight/bias addresses. After their injection, the CNN output vectors (i.e., the vector holding the final predictions) assume fixed values: they all become 0xFFFF (BE15) or 0x0000 (BE23). As a result, the CNN is not able to make correct predictions anymore. The other two block errors (BE5 and BE29) are also categorized as critical since, after their occurrence, all network predictions become wrong too. However, we do not observe specific output patterns (as for BE15 and BE23). The output vectors after the BEs injection

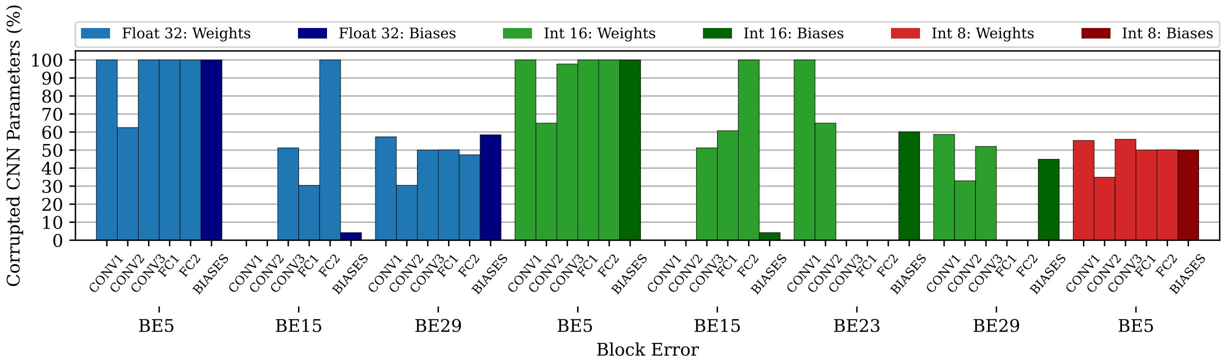

Fig. 12. Investigating the incidence of the most critical block errors on CNNs weights and biases.

assume random values. Moreover, also BE4, when injected, degrades the CNN performance (as illustrated in Figure 11). However, BE4 injects stuck-at-0, and the CNN can still yield some correct predictions after its insertion.

Next, concerning the *Int 8* CNN application, the most critical BE is BE5: when it is injected, the network starts predicting wrong results. More specifically, after its injection, the output vectors of the CNN take on random values. Generally, this CNN is very robust against the occurrence of block errors. Indeed, as shown in Figure 11, only BE23 and BE29 show a slight average degradation in the final accuracy value. The remaining BEs, despite the injected stuck-at bits, do not affect the CNN performance.

While for *Float 32*, it was straightforward to identify the most worrying problem underlying the wrong executions (i.e., stuck-at-1s in the exponent part of the floating-point values), for *Int 16* and *Int 8* it required further in-depth analysis. To this end, we performed a thorough study to investigate the impact of BEs on layers and parameters of the given CNN. As illustrated in Figure 12, for the most critical BEs, the number of corrupted weights or biases (those matching the physical addresses of the BE) is plotted for every layer and for the total biases of the three CNNs. For the sake of clarity, the BEs classified as critical but having an incidence (%) lower than 10% are not included in the plot (i.e., BE18, BE27, BE30, BE37 for *Float 32*).

Depending on the injected block error, the CNN behaviour can be classified in three ways. Once a critical BE occurs in the CNNs, all the output vectors holding the CNN predictions may assume:

- *Random Values:* The percentage of corrupted weights or biases is less than 100% in all layers and biases. Therefore the network continues to propagate values that are not correct but which still depend on inputs (and vary accordingly). This class covers BE5 in *Int 8* and BE29 in *Int 16*. This particular behaviour is observed during the radiation tests of the *Int 16* CNN, specifically for the faulty run 8 in Table 2. The log of the radiation experiment shows, starting from the image number 720, the vector outputs returning all random values (leading to wrong predictions).
- *Fixed Random Values:* In one or more layers the percentage of corrupted weights or biases is 100%. In this case, two scenarios emerge. First, if the block error injects a stuck-at-0 in all the bits of the weights of a layer (BE23 *Int 16*, CONV1), the output of the network no longer depends on the inputs but only on the values assumed by the biases (corrupted at 60% with stuck-at-0s). After the injection of BE23 with the emulator, we observed all 0x0s in all the subsequent output vectors. This outcome is completely matching the faulty run number 6 (Table 2) obtained during the radiation test of the *Int 16* CNN. The report log coming from the radiation test returned all 0x0. Second, if the BE injects stuck-at-1s into all weights of a layer, in the case of integers, the numbers all become negative (since the sign bit becomes equal to 1). Since we use the rectified linear unit (ReLU) as the activation function between layers (except in the last Linear), the negative values become 0. Once again, we lose the dependence on the inputs: the output vector will depend only on the bias values of the last layer, and it will be fixed for all subsequent inferences. This second scenario covers BE5 and BE15 in *Int 16*. Lastly, the same corrupted behaviour (i.e., fixed random values) was observed during run number 9 (Table 2) captured during the radiation test of the *Float 32* CNN.
- *Saturated Values:* This scenario particularly occurred for the *Float 32* CNN. A stuck-at-1 in the highest bits of the exponent part of the floating-point representation leads to NaN exceptions. The propagated values no longer depend on inputs, and the output vectors are fixed to NaN values for all subsequent inferences. This category covers all the critical BEs for *Float 32*.

Based on this classification, the reader could deduce that the injection of BE29 in *Float 32* yields *random* predictions and, similarly, the occurrence of BE5 and BE29 in *Float 32* produces *fixed random* values. However, the problem lies in the corrupted bit position: they inject stuck-at-1s in all bits of each affected weight or bias. As a result, they generate NaN exceptions.

Furthermore, we would like to put a special emphasis on the role of the biases and their resilience against block errors. In the literature, it is claimed that biases are more resilient than weights against statistically injected transient and stuck-at faults [17]. Our experiments show a lower resilience level against block errors. If they are massively affected by faults, the network computation is greatly influenced. For example, BE5 (*Int 16*) corrupts the 100% of total biases parameters by

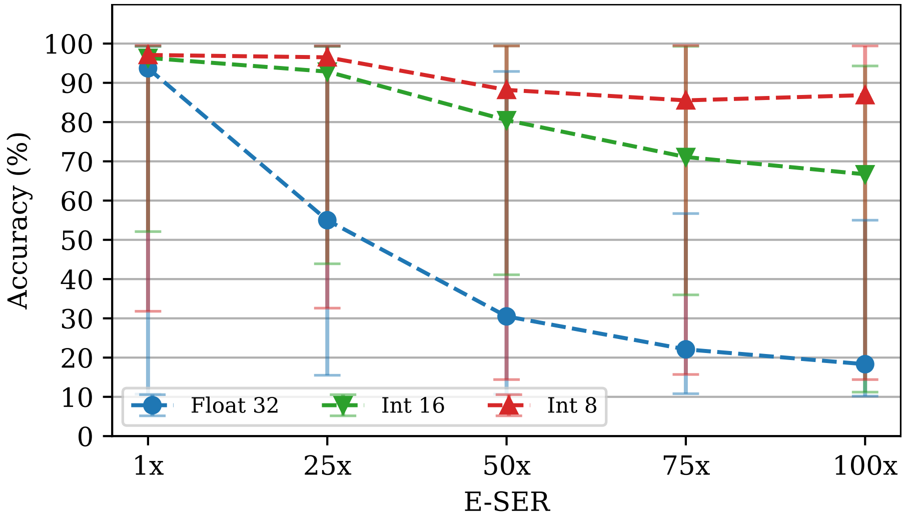

Fig. 13. Accuracy variation based on the increase of the block errors E-SER for the three CNNs.

injecting a stuck-at-1 in the 36% of bits of the matching biases in all the CNN layers. It means that the neuronal computation is greatly upset and corrupted in *all* layers. With the 100% of corrupted biases, the computations turn into *unbiased* values. Furthermore, we experimentally find out that, in the case of BE5 (*Int 8*) and BE29 (*Int 16*), their corruption leads to 4.1% and 3.8% more wrong predictions, respectively, and biases represent only the 0.38% of the total CNN parameters.

Overall, as for non-critical BEs, the final accuracy can slightly decrease but also increase, as highlighted in Figure 11 with the error bars. The same outcome was noticed during the radiation tests, specifically for the faulty run number 7 (the final accuracy slightly decreased), and the faulty run number 10 (moderately increased), as depicted in Table 2.

It is worth pointing out that the above analyses cannot be done only using physical-based injection and, at the same time, a pure software-based fault injection may lead to not significant results if the injected fault models do not reflect the real hardware faulty behaviour. Indeed, knowing which are the faulty components responsible for the observed faulty behaviour makes possible the design and implementation of specific (and thus low cost) fault-tolerant mechanisms.

So far, the effects on the CNN applications of single block errors occurring during a run were addressed. The next analysis is related to the block errors E-SER presented in Table 3 for the 1,000 images dataset. We aimed at evaluating the CNNs resilience when a growing number of BEs affected the CNN parameters (weights and biases). In Table 6 we report the details of the injection campaign, specifically the amount of BEs that are injected in a single run. The reader should note that BEs are not accumulated over the runs, albeit being stuck-at a value.

TABLE 6
Details of block error injections for the CNNs *Float 32*, *Int 16*, *Int 8* with an increasing E-SER: 1x, 25x, 50x, 75x, 100x

E-SER	Total Runs	CNN		
		Float 32	Int 16	Int 8
1x	50	0.32	0.17	0.1
25x	50	8	4	3
50x	50	16	9	5
75x	50	24	13	8
100x	50	32	17	10

First, the emulator was set up with the nominal E-SER (1x) for every CNN application. Then, the E-SER was incremented by 25 times (25x), up to 100 times (100x). Experimental results are shown in Figure 13. For each point on the x-axis, the run was repeated 50 times, with 50 different random sorts of BEs and various random times. The y-axis reports the average final accuracy along with the error bars (depicting the minimum and the maximum value reached). The x-axis reports the increasing E-SER, for which the amount of injected BEs is specified in Table 6.

Experimental results from Figure 13 suggest that depending on either the injection time, the BE sorting or the amount of BEs, the final accuracy can consistently vary, with a non-negligible difference between the maximum and the minimum error bar. In particular, it was observed that the minimum accuracy value was always reached when the first injected BE belonged to the above-described set of the most critical BEs for all the CNNs.

Moreover, the graph shows that the final average decreases as the E-SER increases, in other words, as the number of BEs occurring in a run increases in the three CNNs. It is worth noting that, once again, the results show a greater resilience for the *Int 16* and, above all, *Int 8* CNNs when the number of BEs grows. This is in line with the outcome of the radiation tests.

Finally, it is worth pointing out the following considerations. So far, an enormous effort has been spent in the literature to investigate the reliability of CNNs against transient and permanent faults (e.g., SBUs and stuck-at). At the same time, it must be said that the single fault assumption is not totally realistic. CNNs, like many other applications, mainly suffer the occurrence of multiple faults. However, the establishment of a proper multiple fault injection strategy is still an open challenge and is not a trivial task: the combinations of fault locations can exponentially explode. Therefore, we believe that the block errors injection can be considered as a first realistic attempt to cover multiple faults based on radiation tests results.

5 Conclusion

In this paper, a comprehensive framework to assess the reliability of CNN applications running in embedded devices is presented. The proposed flow starts with a study on the radiation-induced soft errors in CNN applications. Radiation experiments were carried out on the commercial DRAM used for storing the neural network parameters. Through characterisation tests on the system, three types of faults were identified: SBUs, stuck-at bits and block errors. Based on that, we have extracted fault models and event rates to feed a software fault injector able to inject real faults with a realistic frequency. The proposed software emulator is capable of reproducing and injecting the identified fault models on CNN applications.

To demonstrate the validity and efficacy of the emulator, we have used the same CNNs during software emulation and actual radiation tests (*Float 32*, *Int 16*, and *Int 8*), taking care to introduce the errors in the very same portion of data as in the radiation experiments (i.e., *.rodata*). Software FI campaigns were executed on the three CNNs and by exploiting the three types of faults. Experimental results revealed that, in line with the outcomes of the radiation tests, the least resilient CNN application is the one adopting the floating-point data representation; the most resilient is the one using 8-bit integer data type.

We demonstrated that, generally, the software emulator is able to reproduce the faulty behaviours observed during the radiation tests for all three CNNs. For example, by injecting the most critical block errors, we got the same output vectors in both radiation- and software-based FI campaigns, such as NaN for *Float 32* CNN, or all zeros for *Int 16* CNN. Furthermore, we have investigated the reasons for the faulty behavior with the software emulator, by examining the incidence of faults on each individual layer and the distribution between weights and biases. Interestingly, contrary to what is stated in the literature about the biases robustness, our experimental results show a low resilience level against block errors. When biases are massively affected by multiple-bit faults, the CNNs accuracy considerably degrades, and the computations turn into unbiased random values.

The greatest advantage of the proposed software emulator consists in the possibility of injecting realistic fault models. Indeed, despite that radiation-based FIs provide precise reliability assessments, they are extremely costly. Based on the findings in this study, we advocate that this could be complementary to physical testing and may allow optimizing time and costs. Notwithstanding, since the aim of the article is to present this idea, one drawback could be the adoption of three neural networks all based on a single architecture, i.e., LeNet-5. The principal reasons are related to radiation costs and the limited storage capacity of the targeted embedded device. Nevertheless, it is important to underline that one of the goals of the work is to investigate the reliability of different data representations, therefore the need to use only one architecture. In the future, we plan to use the software emulator to assess the reliability of deeper CNN applications with different memory sections and data, as well as other types of memory devices.

Acknowledgments

This study has been achieved thanks to the financial support of the VAN ALLEN Foundation (Contract No. UM 181387) and the Region Occitanie (Contract No. UM 181386), by the European Union's Horizon 2020 research, innovation programme under the Marie Skłodowska-Curie grant agreement No 721624, and by the project "IDEX Lyon OdeLe". The authors would also like to thank Alessandro Ianne for his contribution.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 1026–1034, doi: 10.1109/ICCV.2015.123.

[2] F. F. dos Santos, L. Draghetti, L. Weigel, L. Carro, P. Navaux, and P. Rech, "Evaluation and mitigation of soft-errors in neural network-based object detection in three GPU architectures," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, Aug. 2017, pp. 169–176, doi: 10.1109/DSN-W.2017.47.

[3] F. Libano, B. Wilson, M. Wirthlin, P. Rech, and J. Brunhaver, "Understanding the impact of quantization, accuracy, and radiation on the reliability of convolutional neural networks on FPGAs," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1478–1484, Jul. 2020, doi: 10.1109/TNS.2020.2983662.

[4] G. Di Natale, D. Gizopoulos, S. Di Carlo, A. Bosio, and R. Canal, Eds., *Cross-Layer Reliability of Computing Systems*. Institution of Engineering and Technology, 2020, doi: 10.1049/pbcs057e.

[5] S. E. Damkjar, I. R. Mann, and D. G. Elliott, "Proton beam testing of SEU sensitivity of M430FR5989SRGCREP, EFM32GG11B820F2048, AT32UC3C0512C, and m2s010 microcontrollers in low-earth orbit," in *2020 IEEE Radiation Effects Data Workshop (in conjunction with 2020 NSREC)*, Santa Fe, NM, USA, Dec. 2020, pp. 1–5, doi: 10.1109/REDW51883.2020.9325842.

[6] M. Peña-Fernandez, A. Lindoso, L. Entrena, and M. Garcia-Valderas, "The use of microprocessor trace infrastructures for radiation-induced fault diagnosis," *IEEE Transactions on Nuclear Science*, vol. 67, no. 1, pp. 126–134, Nov. 2020, doi: 10.1109/TNS.2019.2956204.

[7] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, “Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field,” in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Sept. 2015, pp. 415–426, doi: 10.1109/DSN.2015.57.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Dec. 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[9] A. Lotfi *et al.*, “Resiliency of automotive object detection networks on GPU architectures,” in *2019 IEEE International Test Conference (ITC)*, Nov. 2019, pp. 1–9, doi: 10.1109/ITC44170.2019.9000150.

[10] G. Li *et al.*, “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*, Nov. 2017, pp. 1–12, doi: 10.1145/3126908.3126964.

[11] Tiny-CNN. (2016) [Online]. Available: https://github.com/nyanp/tiny-cnn.

[12] A. Ruospo, A. Bosio, A. Ianne, and E. Sanchez, “Evaluating convolutional neural networks reliability depending on their data representation,” in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, Aug. 2020, pp. 672–679, doi: 10.1109/DSD51259.2020.00109.

[13] 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)*, Mar. 2019, pp. 1–6, doi: 10.1109/LATW.2019.8704548.

[14] A. Ruospo, A. Balaara, A. Bosio, and E. Sanchez, “A pipelined multi-level fault injector for deep neural networks,” in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct. 2020, pp. 1–6, doi: 10.1109/DFT50435.2020.9250866.

[15] B. Salami, O. Unsal, and A. Cristal, “On the resilience of RTL NN accelerators: Fault characterization and mitigation,” in *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Sept. 2018, pp. 322–329, doi: 10.1109/CAHPC.2018.8645906.

[16] Y. He, P. Balaprakash, and Y. Li, “Fidelity: Efficient resilience analysis framework for deep learning accelerators,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Athens, Greece: IEEE, 2020, pp. 270–281, doi: 10.1109/MICRO50266.2020.00033. [Online]. Available: https://doi.org/10.1109/MICRO50266.2020.00033

[17] B. Reagen *et al.*, “Ares: A framework for quantifying the resilience of deep neural networks,” in *Proceedings of the 55th Annual Design Automation Conference*. San Francisco, California, USA: Association for Computing Machinery, 2018, pp. 1–6. [Online]. Available: https://doi.org/10.1145/3195970.3195997

[18] D. Xu *et al.*, “Reliability evaluation and analysis of fpga-based neural network acceleration system,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 3, pp. 472–484, 2021, doi: 10.1109/TVLSI.2020.3046075.

[19] G. Tsiligiannis *et al.*, “SEE response evaluation of robotic systems for the nuclear decommissionning,” Mar. 2020, doi: 10.5286/ISIS.E.RB2010438.

[20] L. Dilillo *et al.*, “Evaluation of a fault-tolerant risc-v,” *STFC ISIS Neutron and Muon Source*, Nov. 2020, doi: 10.5286/ISIS E.RB2010053.

[21] C. Cazzaniga and C. D. Frost, “Progress of the Scientific Commissioning of a fast neutron beamline for Chip Irradiation,” *Journal of Physics: Conference Series*, vol. 1021, p. 012037, May. 2018, doi: 10.1088/1742-6596/1021/1/012037.

[22] L. Matana Luza *et al.*, “Investigating the impact of radiation-induced soft errors on the reliability of approximate computing systems,” in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct. 2020, pp. 1–6, doi: 10.1109/DFT50435.2020.9250865.

[23] L. Matana Luza *et al.*, “Effects of thermal neutron irradiation on a Self-Refresh DRAM,” in *IEEE 15th International Conference on Design & Technology of Integrated Systems in Nanoscale Era*, Apr. 2020, pp. 1–6, doi: 10.1109/DTIS48698.2020.9080918.

[24] A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, and A. Virazel, *Advanced Test Methods for SRAMs*. Boston, MA: Springer US, 2010, doi: 10.1007/978-1-4419-0938-1.

[25] D. Niggemeyer, M. Redeker, and J. Otterstedt, “Integration of non-classical faults in standard March tests,” in *Proceedings. International Workshop on Memory Technology, Design and Testing (Cat. No.98TB100236)*, Aug. 1998, pp. 91–96, doi: 10.1109/MTDT.1998.705953.

[26] G. Tsiligiannis *et al.*, “Dynamic test methods for COTS SRAMs,” *IEEE Transactions on Nuclear Science*, vol. 61, no. 6, pp. 3095–3102, Dec. 2014, doi: 10.1109/TNS.2014.2363123.

[27] D. Söderström *et al.*, “Electron-induced upsets and stuck bits in SDRAMs in the jovian environment,” *IEEE Transactions on Nuclear Science*, pp. 1–1, 2021, doi: 10.1109/TNS.2021.3068186. [Early access].

[28] E. Petersen, *Single Event Effects in Aerospace*. Hoboken, NJ, USA: John Wiley & Sons, 2011, doi: 10.1002/9781118084328.

[29] *Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices*, JEDEC Solid State Technology Association Std. 89A, Oct. 2006, [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd-89a.

[30] G. Tsiligiannis *et al.*, “SRAM soft error rate evaluation under atmospheric neutron radiation and PVT variations,” in *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*, Jul. 2013, pp. 145–150, doi: 10.1109/IOLTS.2013.6604066.

[31] C. Torres-Huitzil and B. Girau, “Fault and error tolerance in neural networks: A review,” *IEEE Access*, vol. 5, pp. 17 322–17 341, 2017, doi: 10.1109/ACCESS.2017.2742698.

[32] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998, doi: 10.1109/5.726791.

[33] CEA-LIST. N2D2. [Online]. Available: https://github.com/CEA-LIST/N2D2.

[34] “PG036 - Soft Error Mitigation Controller v4.1 Product Guide,” Xilinx, 2018, [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/sem/v4_1/pg036_sem.pdf.

[35] L. D. Pyeatt, *Modern Assembly Language Programming with the ARM Processor*. Newnes, 2016, doi: 10.1016/C2015-0-00180-0.

[36] R. C. Baumann, “Soft errors in advanced semiconductor devices-part i: the three radiation sources,” *IEEE Transactions on Device and Materials Reliability*, vol. 1, no. 1, pp. 17–22, 2001, DOI: 10.1109/7298.946456.

[37] J. L. Leray, “Effects of atmospheric neutrons on devices, at sea level and in avionics embedded systems,” *Microelectronics Reliability*, vol. 47, no. 9, pp. 1827–1835, 2007, DOI: 10.1016/j.microrel.2007.07.101.

[38] M. F. L’Annunziata, “1 - nuclear radiation, its interaction with matter and radioisotope decay,” in *Handbook of Radioactivity Analysis (Second Edition)*, 2nd ed., M. F. L’Annunziata, Ed. San Diego: Academic Press, 2003, pp. 1–121, DOI: 10.1016/B978-012436603-9/50006-5.