Applying lightweight soft error mitigation techniques to embedded mixed precision deep neural networks

Applying Lightweight Soft Error Mitigation Techniques to Embedded Mixed Precision Deep Neural Networks

Geancarlo Abich, *Student Member, IEEE,* Jonas Gava, *Student Member, IEEE,* Rafael Garibotti, *Member, IEEE,* Ricardo Reis, *Senior Member, IEEE,* and Luciano Ost, *Member, IEEE*

***Abstract*—Deep neural networks (DNNs) are being incorporated in resource-constrained IoT devices, which typically rely on reduced memory footprint and low-performance processors. While DNNs' precision and performance can vary and are essential, it is also vital to deploy trained models that provide high reliability at low cost. To achieve an unyielding reliability and safety level, it is imperative to provide electronic computing systems with appropriate mechanisms to tackle soft errors. This paper, therefore, investigates the relationship between soft errors and model accuracy. In this regard, an extensive soft error assessment of the MobileNet model is conducted considering precision bitwidth variations (2, 4, and 8 bits) running on an Arm Cortex-M processor. In addition, this work promotes the use of a register allocation technique (RAT) that allocates the critical DNN function/layer to a pool of specific general-purpose processor registers. Results obtained from more than 4.5 million fault injections show that RAT gives the best relative performance, memory utilization, and soft error reliability trade-offs w.r.t. a more traditional replication-based approach. Results also show that the MobileNet soft error reliability varies depending on the precision bitwidth of its convolutional layers.**

***Index Terms*—Soft Error, Reliability, Machine Learning, IoT.**

I. Introduction

More recently, there has been an expedited trend in incorporating deep neural networks (DNNs), in particular the convolutional ones, in resource-constraint Internet of Things (IoT) devices [1], [2]. To enable DNN models' execution on the underlying devices, software libraries and application programming interfaces (APIs) have been proposed [3], [4], [5]. Such libraries/APIs are devoted to streamlining the design and development of embedded deep learning-based applications through the fine-tuning of pre-trained network models, thus enabling their efficient execution in edge-computing platforms [2], [6]. For the time being, the majority of embedded trained models and their inference engines have been evaluated only according to their accuracy and performance over a given dataset.

With the growing adoption of DNNs in safety-critical embedded systems (e.g., medical devices, autonomous vehicles), increases the demand for safe and reliable models. To reach levels of reliability that are comparable to those required by high safety standards [7], it is imperative to supply electronic computing systems with appropriate mechanisms to reduce their vulnerability to radiation-induced soft errors. This work advocates that traditional redundancy mitigation approaches may not be suitable templates for tackling the occurrence of soft errors in IoT edge devices. The underlying approaches, generally, incur high performance and memory overheads, making them impractical to be deployed in resource-constrained systems. The resulting scenario poses two challenging questions: (*i*) which is the relationship between the soft error susceptibility and model accuracy?, and (*ii*) how to reduce the risk of radiation-induced soft errors in DNNs executing on resource-constrained devices? An initial attempt to identify the relationship between the soft error susceptibility and model accuracy was conducted in [8], where authors show that MobileNet convolutional neural network (CNN) with higher precision bitwidth configurations led to a higher number of soft errors. This work only considers 4 and 8 bits per-layer (PL) compression. On the soft error mitigation side, traditional techniques have been either implemented in FPGA [9] or applied to DNN accelerators [10], which benefit from substantial computational parallelism w.r.t. microprocessors.

To address the aforementioned challenges, this paper *contributes* by assessing the impact of precision bitwidth on the soft error reliability of the MobileNet CNN [11] when running on an Arm Cortex-M7 processor. It is the first work to consider weights and activations quantization at 2, 4, and 8 bits while applying the per-channel (PC) compression and integer-channel normalization activation (ICN) technique. The other contributions of this work are as follows:

- Promote the use of RAT [12], a lightweight soft error mitigation technique, as an effective alternative to the traditional replication techniques, which have a reasonable impact on the resource-constraint system's performance and response time;
- Extensive soft error assessment of MobileNet CNN on ImageNet considering more than 4.5 million fault injections;
- Relative performance, memory utilization, and soft error reliability trade-offs analysis considering RAT and a partial triple modular redundancy (P-TMR);
- Hardened MobileNet source code execution in a real board.

G. Abich, J. Gava and R. Reis are with PGMicro, UFRGS, Brazil (e-mail: gabich, jfgava, reis@inf.ufrgs.br).

R. Garibotti is with School of Technology, PUCRS, Brazil (e-mail: rafael.garibotti@pucrs.br).

Corresponding author: L. Ost is with Loughborough University, UK (e-mail: l.ost@lboro.ac.uk).

Manuscript received April 15, 2021.

The rest of this paper is organized as follows. Section II presents the related works in machine learning algorithms soft error assessment and mitigation, considering different approaches. Section III details the adopted fault injection framework, along with the description of the fault classification, the mitigation techniques, and the evaluation metrics used in this work. Section IV presents our case study, detailing both MobileNet CNN and the CMix-NN library. Following, Section V explores the soft error reliability of the MobileNet CNN considering different aspects: precision bitwidth, layer vulnerability, mitigation techniques, and relative trade-off analysis. Finally, Section VI points out conclusions and future work.

II. Related Work in Machine Learning Soft Error Assessment and Mitigation

The soft error assessment and mitigation literature is abundant, requiring a taxonomy to classify the different approaches. This proposal considers the definitions from [13] for fault, error, and failure. A fault is an event that may cause the internal state of the system to change, e.g., a radiation particle strike. When a fault affects the system's internal state, it becomes an error. If the error causes a deviation of at least one of the system's external states, then it is considered as a failure. To achieve compliance with safety and reliability standard requirements, it is utmost importance to provide systems with appropriate mechanisms to tackle systematic or transient faults, also known as soft errors or Single Event Upset (SEU). While the former originates from hardware and software design defects, soft errors are those caused by alpha particles or atmospheric neutrons [14].

The occurrence of soft errors problem can be tackled both in hardware and software. While hardware approaches lead to the area and power overhead, software techniques are generally implemented on a per-application basis that usually incurs performance penalties. The following Section reviews system-level soft error techniques rather than technology-specific approaches that required control of the chip fabrication process, which is often outsourced.

A. Review of System-level Soft Error Mitigation Techniques

Nicolescu *et al.* [15] propose an error detection technique that is based on the introduction of data and code redundancy using a set of transformation rules applied to high-level code. In turn, Benso *et al.* [16] introduce the REliable Code COmpiler (RECCO), a tool that exploits code reordering and selective variable duplication to generate hardened C/C++ source code automatically. Serrano-Cases *et al.* [17] use genetic algorithms to find a combination of optimization flags that can increase the final binary reliability while maintaining a reasonable performance and memory utilization trade-off. Rodrigues *et al.* [18] developed software TMR and Conditional Modular Redundancy (CMR) mitigation implementations, aiming to reduce the occurrence of soft errors in a Cortex-A9 processor running Linux kernel.

Another software-based alternative to mitigate soft errors comes from low-level code protection. Authors in [19], promote the SWIFT (SoftWare Implemented Fault Tolerance) technique aiming to reduce the overhead associated with EDDI (Error Detection by Duplicated Instructions) [20]. They remove duplicate store instructions, reducing both memory and performance overhead. The SWIFT technique assumes that the system's memory architecture is protected by some error correction mechanism. Results showed a 14% speed-up over EDDI when tested with an Intel Itanium 2. Authors in [21] improved SWIFT technique by checking the load instructions right after a store instruction and creating redundant load instructions in critical sections to achieve near-zero the occurrence of silent data corruption (SDC). A popular instruction-level mitigation technique introduced by Reis et. al [22] is the SWIFT-R, which implements TMR to recover from soft errors in the register file. Instead of duplicating instructions, it triplicates, and change the checking points to a voter mechanism.

In [23], authors presented the Shoestring technique, which exploits a low-cost symptom-based error detection mechanism that focuses on applying instruction duplication to protect only those code segments that are likely to result in user-visible faults and do not exhibit symptomatic behaviour. Results show that Shoestring can recover from an additional 33.9% of soft errors that are undetected by a symptom-only approach. [24] presents the Encore, a software-based error recovery mechanism (paired with other error detection techniques) that combines program analysis, profile data, and simple code transformations to create code portions, which can recover from faults at a minimal cost. Gathered results show that Encore can recover from 97% of transient faults on average with 14% additional runtime overhead. Another TMR-based technique, called ELZAR, is proposed in [25]. It triplicates arithmetic and logical operations, and the voting mechanisms are inserted between register operands of memory and control flow operations for recovery. To reduce the performance overhead introduced by replicated instructions, they utilize Intel AVX extensions (i.e., Single Instruction Multiple Data - SIMD). The experiments show that the performance overhead is reasonable for CPU-intense applications with many floating-point operations. However, for some case studies, the instruction-level parallelism was inefficient, resulting in a performance penalty that surpassed the SWIFT-R technique.

The NEMESIS technique introduced by [26] is a duplication with recovery technique. It replicates instructions and checks the results of memory write operations and branches' direction. If an error is detected, it then recovers to a valid state if possible; otherwise, a power restart is needed. The results show that at least 97% of the detected errors were recoverable considering the ten selected applications. Another error recovery technique is the InCheck [27], which is an extension of the nZDC technique [21]. The proposed technique comprises of error detection, diagnosis, and recovery schemes. Unlike SWIFT-R, the InCheck mitigates faults by protecting error handling routines in addition to the main program instructions. The authors claim that their technique offers complete error coverage for the tested applications.

TABLE I
RELATED WORKS IN MACHINE LEARNING ALGORITHMS SOFT ERROR ASSESSMENT AND MITIGATION.

Work	Dataset	ML Algorithm/Model	FI approach	Target	Mitigation	Description
Li et. al [10] (2017)	ImageNet, CIFAR-10	ConvNet, AlexNet, CaffeNet, NiN	Simulation	DNN accelerator	Symptom-based, Selective Latch Hardening	Assess the propagation of soft errors in five neural nets running on nine DNN accelerators and propose solutions to mitigate the errors' impact.
Libano et. al [28] (2018)	Boston Housing, Iris Flower	2-layer NN	Emulation, Heavy-ion irradiation	Xilinx Zynq-7000		Evaluate the soft error reliability of two different feedforwards ANNs implemented on Xilinx Artix-7 FPGA. Analyse the criticality of errors in distinct layers of the ANNs and evaluate the activation function complexity's impact on the neural network reliability.
Reagen et. al [29] (2018)	MNIST, CIFAR-10, ImageNet, TIDIGITS	Fully connected, CNN, gated recurrent unit (GRU)	Simulation, Emulation	DNN accelerator		Propose Ares, a framework capable of assessing the fault tolerance and accuracy trade-offs of three types of DNNs (fully connected, CNN, and GRU) at different levels (model, layer, structure).
Santos et. al [30] (2018)	PASCAL VOC, Caltech Pedestrian	YOLO, Faster R-CNN, ResNet	Simulation, Neutron irradiation	NVIDIA GPU	ABFT	Evaluate the reliability on three GPU platforms executing three neural networks and investigate the effectiveness of ABFT technique in the mitigation of soft errors.
Rosa et. al [31] (2019)	KITTI Visual Odometry	CNN	Simulation	Arm Cortex-A9		Present a soft error assessment flow that enables the identification of soft errors in complex software stacks (e.g., autonomous vehicles) and determine the correlation between multicore platform microarchitectural characteristics and detected soft errors using ML techniques.
Trindade et. al [32] (2019)	Fault Detection	Support Vector Machine (SVM)	Neutron irradiation	Xilinx Zynq-7000		Assess the intrinsic fault tolerance of an SVM architecture implemented on Artix-7 FPGA identifying the exact nodes that are more likely to conduct critical failures.
Libano et. al [9] (2019)	Iris Flower, MNIST	2-layer NN, 7-layer CNN	Emulation, Neutron irradiation	Xilinx Zynq-7000, Zynq Ultrascale+	Partial TMR, Full TMR	Assess the radiation-induced errors in two neural networks (a simple two-layer ANN and a larger seven-layer CNN) implemented on two distinct Xilinx platforms and compared them with the Full TMR and Partial TMR versions.
Chen et. al [33] (2019)	MNIST, Survive, Cifar-10, ImageNet, German traffic sign, Driving	2-layer NN, LeNet-4, kNN, AlexNet, VGG16, VGG11, Nvidia Dave, Comma.ai	Simulation	Intel X86-64 processors		Propose a python-based fault injector, called BinFI, to analyse the safety-critical bits in ML-based applications with significant lower costs comparing to random FI.
Trindade et. al [34] (2020)	Iris Flower	ANN, SVM	Emulation, Neutron irradiation	Arm Cortex-M4		Assess the radiation-induced soft error reliability of two ML algorithms (ANN and SVM) running on a low-power Arm Cortex-M4 processor using the STM32 NUCLEOL45RE-P platform.
Luza et. al [35] (2020)	MNIST	LeNet-5 CNN	Neutron irradiation	Xilinx Zynq-7000		Assess the impact of radiation-induced soft errors on HyperRAM memory containing the CNN application weights and input data. The work utilises three distinct data representations (32-bit float, 16-bit and 8-bit integer), and experiments show that when the bitwidth is reduced, the soft error resiliency also decreases.
Abich et. al [36] (2020)	CIFAR-10	7-layer CNN	Simulation	Arm Cortex-M3, Arm Cortex-M4		Assess the layer soft error reliability of a CNN model implemented with the CMSIS-NN kernels considering two resource-constrained Arm processor models.
Kundu et. al [37] (2021)	MNIST, Fashion-MNIST	Multilayer Perceptron, LeNet-5, AlexNet, VGG16, ResNet-50	Emulation	DNN accelerator		Assess the faults in the accelerator's memory subsystem running a Multilayer Perceptron (MLP), analyse MAC circuits' resilience when faults strike the MSB and LSB logic cone, and finally propose mitigation solutions for aging, thermal and endurance issues on neuromorphic hardware.
This work (2021)	**ImageNet**	**MobileNet CNN**	**Simulation**	**Arm Cortex-M7**	**Partial TMR, RAT**	Assess the soft error reliability of the MobileNet CNN running on the resource-constrained Arm Cortex-M7 processor considering mixed-precision bitwidth configurations. Evaluate the soft error impact on system reliability when different lightweight software-based mitigation techniques are applied.

B. Review of Soft Error Assessment of ML algorithms

The number of products integrating Machine Learning (ML) algorithms is continuously increasing. With this in mind, researchers have started to investigate the impact of radiation-induced soft errors on the reliability of such algorithms, as summarized in Table I.

In the context of soft error assessment, with the exception of [34], [36] and this work, reviewed approaches do not consider resource-constraint on their experiments. The majority of these works consider either FPGA implementations of ML algorithms [9], [32], [35] or their execution on GPU [30], DNN accelerators [10], [29], [37] or general-purpose processors [31], [33]. On the soft error mitigation side, traditional partial TMR or specific mitigation techniques have been considered either in FPGA implementations [9] or applied to specialized hardware accelerator [10] or more generic GPUs [30].

Li et al. [10] assess the soft error resilience of DNNs running on specific accelerators. In this work, the authors conducted an in-depth study of the applications' functioning while promoting bespoke mitigation solutions to each case seeking a lower cost. Different from this work, our approach

facilitates the application of mitigation techniques through a more generic and automated approach to protect the most critical functions/layers of generic or ML-based applications, considering the possibility of manual configurations to guide bespoke hardening tuning for resource-constraint devices. Authors in [28], [32] assess the soft error reliability of distinct ML algorithms implemented in FPGA devices, while Libano et al. [9] increment the evaluation by presenting full and partial hardware replication solutions to recover from errors.

Santos et al. [30] focus on Graphics Processing Units (GPUs), which are widely used in high-computational systems. In this work, the Algorithm-Based Fault Tolerance (ABFT) technique has been applied due to its efficient to detect soft errors in dense linear algebra operations including matrix multiplication, which is highly performed by CNNs. Although ABFT brings less performance overhead w.r.t. replication-like approaches (e.g., TMR), resulting overhead might still lead to high response times - gold criteria to resource-constrained devices.

This paper extends from previous work in three key directions:

- First, this is the first work to investigate the relationship between the soft error susceptibility and model accuracy, which is completely ignored in the works presented in Table I;
- Second, our work puts focus on reducing the occurrence of soft errors in resource-constraint devices. Therefore, this is the first work to evaluate the benefits of using a lightweight technique (e.g., RAT) w.r.t. a partial replication technique;
- Third, this work explores the relative performance, memory utilization, and soft error reliability trade-offs of two system-level mitigation techniques considering a microprocessor running different precision bitwidth variations of MobileNet on ImageNet.

III. Adopted Fault Injection Framework

Rather than developing a fault injection (FI) framework from scratch, this work adopts the SOFIA soft error assessment flow [38]. This choice is justified because SOFIA provides a set of well-accepted FI techniques along with several facilities (e.g., error tracer module), which allows to identify and classify the effects of soft errors on the system's behaviour, considering both hardware and software architectures. This Section covers all relevant steps used to assess the impact of precision bitwidth on the soft error reliability of MobileNet CNN, including the description of the adopted profiling, the fault classification, the mitigation techniques, and the evaluation metrics used in this work.

A. Profiling and Soft Error Assessment

To enable the soft error assessment of emerging ML-based applications, software engineers must be able to execute complex software stacks with hundreds of billions of instructions, early in the design phase. Due to the complexity of such stacks (e.g., kernels, drivers, and applications), analysing their soft error reliability may take several months if conducted at low level simulations (e.g., gate-level). In this regard, this work uses SOFIA [38], a framework based on OVPsim [39], which allows injecting faults at a speed of up to 1000 MIPS while preserving the soft error analysis accuracy (i.e., mismatch below to 10%) for single and multicore processors [40]. SOFIA emulates the occurrence of single-bit upsets (SBUs) by injecting faults into pre-selected data storage elements (i.e., registers and memory addresses) during the execution of a given software stack. The fault injection configuration (e.g., bit location and injection time) relies on a random uniform function, which is a well-accepted fault injection technique since it covers the majority of possible faults on a system at a low computation cost. The framework supports the injection of bit-flips in six different scopes: register file, physical memory, application virtual memory, application variables and data structures, function object code, and function lifespan. These techniques allow covering either architecture aspects as well as isolate specific kernel (e.g., scheduling) or deep inference network function (e.g., matrix multiplication), thus covering both spatial and temporal faults.

This work uses two complementary FI techniques to assess deep inference networks' soft error reliability: *random register file* and *function lifespan*. On the one hand, *random register file* FI is a well-accepted mechanism that homogeneously covers most soft errors, striking the general-purpose registers while both application and operating system codes are executing. On the other hand, *function lifespan* reduces the FI spectrum by limiting the insertion time to those small intervals where the target function is active.

Figure 1 shows the fault injection flow used in this work. First, the *Platform Setup* defines the parameters used in the fault campaign, such as the target architecture and the evaluated application. Then, the *Gold Reference Model* step is performed to extract the execution data without fail, which will generate our faultless reference. *Fault Injection Setup* is another step with human intervention. This step defines the number of bit-flips that will be performed according to a statistical model, thus generating a list of simulation moments and registers' bit where the fault injections will occur. Next, *Fault Injection Simulation* performs fault injection campaigns; at the end of each run, SOFIA extracts the processor's register bank memory dump, in addition to the application output to be used for comparison with the gold reference data. Finally, *Fault Analysis* compares the experimental results, which are grouped according to a fault classification that can be defined by the user.

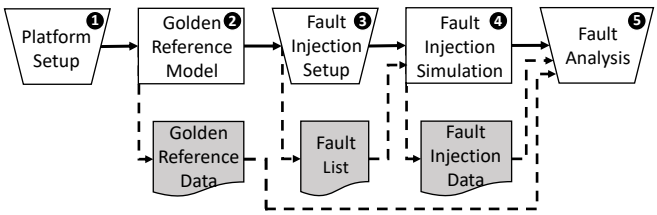


Fig. 1. FI flow: covering from the platform setup to the fault analysis.

The fault classification adopted in this work follows the pattern shown in [10], [41], [34], where the authors identify the

faults in the outputs as: *correct output*, *tolerable faults*, *critical faults*, and *crashes*. *Correct outputs* are those where the output data (e.g., output probability) is the same as the ones obtained from the faultless execution (i.e., golden reference data). In *tolerable faults*, the output data differs from the gold reference data. However, they present a top-ranked classification equals to the fault-free execution. On the other hand, *critical faults* affect the output with incorrect probabilities and no predictions (i.e., cases where odds are dispersed, therefore, they have no probabilities in the output data). Finally, *crash* comprises the application that ends abnormally with an error indication or does not finish, requiring a preemptive removal after a threshold execution time.

Note that the main difference between *critical fault* and *crash* is that the former refers to a silent error (i.e., the application ends without an error signal) while the latter is a detectable one (i.e., an error signal or unexpected behaviour). Silent errors are considered critical in this work as they can be propagated, which might ultimately incur in human life losses for safety-critical applications (e.g., autonomous vehicles). In contrast, detectable errors can be handled by the system as there is the possibility to reset the system or rerun the algorithm to obtain the correct result.

B. Mitigation Techniques

To ensure failsafe functionality of ML-based systems, reliability engineers should be able not only to identify but also explore efficient mitigation solutions to reduce the occurrence of soft errors. This work also aims to improve the soft error reliability through the application of two software-based mitigation techniques: *P-TMR* and *RAT*.

The first mitigation technique is based on a replication approach, i.e., a technique that replicates instructions (except stores and branches) and adds majority voters before conditional branches, load, and store instructions on top of an intermediate representation (i.e., LLVM IR [42]). However, unlike traditional approaches that replicate the entire code, this work uses the *partial TMR* (*P-TMR*) technique that only replicates specific/critical functions, thus minimizing the performance overhead.

The second adopted mitigation technique is the *register allocation technique* (*RAT*) [12]. This hardening technique restricts the number of available registers used to execute specific functions, thus reducing the exposed area. Unlike replication approaches, RAT does not involve code redundancy and is an architecture-independent approach. Also, RAT is a compiler-based technique; thus, it can be associated with other mitigation techniques applied at the LLVM IR level (e.g., our P-TMR technique).

C. Evaluation Metrics

To properly assess the soft error reliability impact on a given system, reliability metrics must be used. This work uses the *mean work to failure* (*MWTF*) metric [43].

Complementary to the fault classification, the MWTF shows the average amount of work that an application can perform until reaching a failure (i.e., higher values are better). This is a fair metric to either compare or evaluate the effects generated by different mitigation techniques. This metric is evaluated in the *Fault Analysis* step (Figure 1). Note that for deep inference networks, the unit work is defined as the relationship between the application's runtime and the most critical vulnerability (i.e., *critical faults*), as shown in Equation (1).

$$MWTF = \frac{1}{(execution\ time \times AVF_{CriticalFaults})} \quad (1)$$

The Architecture Vulnerability Factor (AVF) is used to measure the probability of a fault result in an error (i.e., SDC or Crash) [44]. The AVF critical considers only the SDCs that actually led to wrong classifications in this case study (i.e., critical faults). For example, in safety-critical applications, such as autonomous cars, a critical fault can alter the detection of an obstacle in front of the vehicle, which can lead to an accident. For this reason, this work used the critical-based AVF ($AVF_{critical\ faults}$).

IV. Case Study

This Section describes the MobileNet CNN [11], which was used to investigate the relationship between soft errors and model accuracy. The MobileNet CNN precision was set using the CMix-NN library [5] and executed on an Arm Cortex-M7 processor. The MobileNet is trained with the ImageNet dataset [45], which consists of 10 million labelled images divided into 1000 object classes. To present the adopted CNN application, Section IV-A describes the MobileNet while Section IV-B shows the CMix-NN library.

A. MobileNet CNN

MobileNet CNN [11] is a streamlined architecture that aims to build lightweight deep inference networks. The MobileNet CNN topology consists of several convolution layers composed of depthwise and pointwise convolutions, average pooling layers, and a fully connected layer. In this case study, the adopted MobileNet CNN was configured with a 3×3 depthwise separable convolution, representing savings of up to 9 times in computational cost compared to standard convolutions. After convolution layers, an average pooling reduces the spatial resolution to 1 before the fully connected layer. In this sense, MobileNet has 29 layers, considering depthwise and pointwise convolutions as separate layers, except for the first layer that is a full standard convolution.

B. CMix-NN Library

The adopted MobileNet CNN uses the CMix-NN library [5] to implement mixed low-precision standard convolution and depthwise separable convolution layers functions. While traditional CNN models are trained using 32-bit floating-point data representation, the CMix-NN uses mixed low-precision unsigned integer representation, where the weights and bias are calculated using a custom precision training. The CMix-NN kernels deploy optimizations focusing on enabling the

execution of CNNs on Cortex-M based systems that support *single instruction/multiple data* (SIMD) instructions, especially 16-bit *multiply-and-accumulate* (MAC) instructions (e.g., SMLAD). The CMix-NN library provides a complete set of convolutional kernels featuring a mixed low-bitwidth for the weights, input, and output activations that support 8, 4, and 2 bitwidth combinations and different quantization techniques.

A typical mixed-precision *quantized convolutional layer* (QCL) workload splits the convolution between quantized image-to-column and a matrix multiplication loop. The quantized functions load *Q*-bits input data in temporary buffers casting from the original *Q*-bits format to execute through vectorized SIMD 2×16 MAC instructions. Figure 2 illustrates the QCL internal components with memory requirements (e.g., inputs, weights, and structures) and the computational dataflow, which implement the mixed low-precision convolutional functions. The low-precision *MAC unit* accumulates the convolution result over a temporary 32-bit precision variable through vectorized MAC operations. In asymmetric quantizations, Z_w and Z_i apply the offset to the loaded parameter values to transpose them into the custom asymmetric domain. While the *Unpack* operation loads the convolution operands, the *Compressor* unit operates the final compression on the high-precision accumulation, considering a set of parameters T_A, which varies depending on the applied quantization technique.

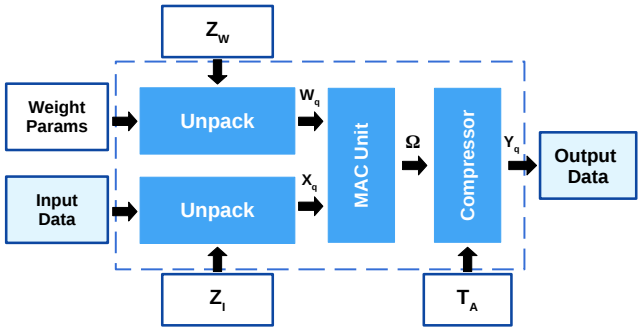


Fig. 2. CMix-NN quantized convolutional layer (adapted from [5]).

In addition, CMix-NN library supports *per-layer* (PL) and *per-channel* (PC) compression techniques for any combination of bitwidth between input, output, and weights. While a PL quantization exploits a single *min/max* value for the entire layer, the PC computes a *min/max* value for any output channel. This latter approach is most beneficial when the weight distribution varies widely between channels. Furthermore, the CMix-NN library also supports the *integer-channel normalization activation* (*ICN*) [46]. This technique allows the introduction of lower bitwidth models with negligible inference loss, opening opportunities to exploit the soft error reliability of convolutional layers with precision bitwidth.

V. Results

This Section explores the soft error reliability of the MobileNet CNN considering different aspects: precision bitwidth, layer vulnerability, mitigation techniques, and relative trade-off analysis. In this sense, Section V-A details the experimental setup used to perform the fault injection campaigns. Section V-B exploits the MobileNet soft error reliability considering different precision bitwidth configurations. Then, Section V-C presents the reduction of soft errors when applying two system-level mitigation techniques. Finally, Section V-D presents the relative performance and reliability trade-off for the adopted soft error mitigation techniques.

A. Experimental Setup

To provide trustworthy results, experiments consider more than 4.5 million fault injections to assess the soft error reliability of two mitigation techniques applied to the MobileNet on ImageNet. This work considers two FI techniques (i.e., *function lifespan* and *random register file*) to inject flipped bits in the general-purpose registers (i.e., r0-r15) of the Arm Cortex-M7 processor. This choice was motivated for two reasons. First, because the two FI techniques provide better coverage of the soft errors presented. Second, because all CMix-NN optimizations, including the SIMD instructions, require only the general-purpose registers. Note that this work focuses on the assessment and mitigation of soft errors originated from general-purpose registers, and hence it is assumed that the memory is protected by some type of error correction, such as ECC or parity bit.

Table II shows the experimental setup. The adopted MobileNet CNN has per-channel quantization with ICN layers (PC+ICN [46]), configured with the width multiplier of 0.5 and input sizes of 192, since this configuration has the minimum channel width required by 2-bit configurations.

TABLE II
Experimental Setup

Processor	Arm Cortex-M7
ML Model	MobileNet CNN
Dataset	ImageNet
Topology	1 Standard Convolution, 13 Depthwise, 13 Pointwise, 1 Average Pooling, 1 Fully-Connected
Target Layers	All, L01, L02, L03, L04, L25, L26, L27, L28, L29
Evaluated Precisions ***(w: weights, a: activations)***	w2a2, w2a4, w2a8, w4a2, w4a4, w4a8, w8a2, w8a4, w8a8
Mitigation Techniques	P-TMR and RAT
Number of FI campaigns	270
Injections per campaign	17k
Total Fault Injections	4.59 millions

In addition, this work uses the same compilation environment (i.e., *Clang 6.0.1* and optimization flag *-O2*) to set the bitwidth configurations and the mitigation techniques. Each fault injection campaign considers a single input image for each MobileNet CNN execution, thus not considering the fault propagation to the subsequent executions. Also, each campaign is a particular configuration scenario and the 270 campaigns comprise: 10 target layers * 9 precision bitwidths * 3 mitigation cases (Code unprotected, P-TMR, and RAT). Furthermore, conducting a precise, well-covered, and realistic

TABLE III
PERCENTAGES OF TOLERABLE AND CRITICAL FAULTS ON MOBILENET CNN CONSIDERING DIFFERENT PRECISION BITWIDTH CONFIGURATIONS.

		w2a2				w2a4				w2a8			
	Layers	Correct	Tolerable	Critical	Crash	Correct	Tolerable	Critical	Crash	Correct	Tolerable	Critical	Crash
1	Standard Convolution	91.65	**0.01**	**2.03**	6.31	89.98	**0.01**	**1.42**	8.58	89.81	**0.41**	0.93	8.85
2	Depthwise	92.66	**0.08**	0.15	7.11	92.77	**0.01**	0.24	6.98	85.62	5.02	**2.95**	6.41
3	Pointwise	90.58	**0.07**	**3.76**	5.58	88.32	**0.12**	**3.44**	8.12	83.31	**2.06**	**6.23**	8.40
4	Depthwise	91.61	**0.06**	**1.55**	6.77	88.55	**0.00**	**4.37**	7.08	79.93	6.59	**7.11**	6.36
25	Pointwise	92.84	**1.31**	0.47	5.39	89.75	**1.85**	0.58	7.82	87.53	**3.66**	0.78	8.03
26	Depthwise	84.49	7.10	**1.29**	7.12	84.44	7.13	**1.26**	7.17	82.22	11.27	0.55	5.96
27	Pointwise	80.04	9.84	**2.13**	8.00	79.94	11.15	**1.21**	7.70	79.28	12.19	0.71	7.81
28	Average Pooling	65.28	20.11	**3.86**	10.74	65.19	20.02	**4.04**	10.75	65.55	20.19	**3.47**	10.78
29	Fully-Connected	75.08	17.57	**1.89**	5.46	74.30	18.15	**1.93**	5.62	74.19	18.17	**1.97**	5.67
all	All	91.32	**0.84**	**1.80**	6.04	88.51	**1.11**	**2.66**	7.72	83.96	5.45	**3.02**	7.56
		w4a2				w4a4				w4a8			
	Layers	Correct	Tolerable	Critical	Crash	Correct	Tolerable	Critical	Crash	Correct	Tolerable	Critical	Crash
1	Standard Convolution	91.81	**0.00**	**1.75**	6.45	90.18	**0.01**	**1.08**	8.74	89.60	**0.60**	0.87	8.93
2	Depthwise	92.84	**0.00**	0.15	7.01	92.79	**0.05**	0.18	6.98	84.63	8.72	0.65	6.00
3	Pointwise	90.64	**0.03**	**3.78**	5.55	88.51	**0.01**	**3.47**	8.01	81.47	6.35	**4.31**	7.86
4	Depthwise	91.60	**0.00**	**1.54**	6.86	88.04	**0.04**	**5.03**	6.89	78.94	13.78	**1.19**	6.08
25	Pointwise	92.11	**2.04**	0.25	5.60	89.95	**1.92**	0.42	7.71	85.76	5.78	0.54	7.93
26	Depthwise	85.49	6.77	**1.21**	6.53	84.36	7.09	**1.50**	7.05	80.51	13.05	0.46	5.98
27	Pointwise	81.51	8.35	**2.32**	7.83	82.10	8.89	**1.43**	7.58	80.22	10.81	0.94	8.04
28	Average Pooling	65.66	19.55	**3.85**	10.94	65.41	20.21	**3.79**	10.59	66.04	19.54	**3.49**	10.94
29	Fully-Connected	75.19	17.18	**1.95**	5.67	74.21	18.16	**1.99**	5.64	74.42	17.91	**2.16**	5.51
all	All	91.79	**0.66**	**1.88**	5.67	88.43	**0.72**	**3.24**	7.61	81.68	8.61	**1.86**	7.85
		w8a2				w8a4				w8a8			
	Layers	Correct	Tolerable	Critical	Crash	Correct	Tolerable	Critical	Crash	Correct	Tolerable	Critical	Crash
1	Standard Convolution	92.88	**0.01**	0.70	6.41	90.19	**0.01**	0.87	8.93	89.78	**0.64**	0.66	8.92
2	Depthwise	92.85	**0.06**	0.19	6.89	92.06	**0.03**	0.32	7.58	83.90	8.90	0.54	6.66
3	Pointwise	90.56	**0.04**	**3.58**	5.82	87.73	**0.02**	**3.87**	8.38	78.65	11.58	**1.49**	8.28
4	Depthwise	91.60	**0.06**	**1.39**	6.95	86.15	**0.02**	**6.35**	7.48	75.77	17.18	0.68	6.36
25	Pointwise	89.99	4.14	0.20	5.68	88.72	**3.23**	0.28	7.77	79.27	9.22	**3.54**	7.97
26	Depthwise	84.19	7.03	**1.62**	7.16	82.93	7.94	**1.92**	7.21	79.98	8.45	**4.94**	6.64
27	Pointwise	81.24	8.51	**2.49**	7.76	81.39	8.86	**1.64**	8.11	84.01	6.58	**1.64**	7.78
28	Average Pooling	65.80	19.73	**3.55**	10.92	65.65	19.56	**3.69**	11.10	66.73	6.89	**15.44**	10.94
29	Fully-Connected	74.52	17.67	**2.13**	5.68	73.56	18.39	**2.11**	5.94	75.79	16.56	**2.23**	5.42
all	All	91.60	**0.74**	**1.78**	5.88	86.84	**1.22**	**3.61**	8.34	78.71	11.64	**2.21**	7.45

approach is key when assessing a system's soft error reliability. In this sense, to ensure the results' statistical significance, this work injects 17k faults per campaign, which according to Leveugle et al. [47], generates a margin of error of 1% with a 99% confidence level.

B. Soft Error Reliability Assessment of the MobileNet CNN Considering the Precision Bitwidth

Initially, we validated the SOFIA framework's faultless reference against the outputs reported by the MobileNet repository[1] and its on-chip execution. In this regard, we executed MobileNet CNN on an STM32H743[2] device, and then compared it with those collected from its execution on SOFIA, where no difference in the output probabilities was shown. This experiment is of paramount importance to guarantee the reproducibility and meaningfulness of the results.

After validating the reference flow, we generate fault injection campaigns considering the variation of precision bitwidth with weights raging from *w2* to *w8* and input/output activations from *a2* to *a8*. Table III shows the MobileNet CNN soft error results detailing the fault classification for each bitwidth configuration.

In general, results show a similar soft error reliability behaviour between the different quantization configurations. For example, the results from fault injections in *All* layers in Table III, crash occurrences handle 6.92% on average across all precision bitwidth configurations; this is because MobileNet CNN functions use multiple loops to process data that require many control instructions. Note that CNNs are known to have a lot of redundancy built-in, due to which they present a reasonable masking rate, which justifies the average of 86.7% of correct outputs. However, a single critical fault occurrence in safety-critical systems running the underlying trained models can lead to fatal consequences (i.e., life-threatening).

Table III also shows how the quantization of inference activations affects the MobileNet CNN soft error reliability by reducing the correct outputs in up to 10.12% when varying from *a2* to *a8*. Although it affects less, increasing the weight bitwidth also reduces the soft error reliability by up to 2.21% on the correct outputs when varying the configurations from *w2* to *w8*. This occurs because a SIMD MAC instruction splits a 32-bits register into different segments, which are set according to the bitwidth configuration. This leads to a higher probability to overwrite the faulty bit, thus reducing the probability to propagate the fault to the inference phases. Results show that both higher precision bitwidths and the unpack/compress process can led to an increase number of faults. The unpack/compress process is related to load and store instructions that are executed before and after SIMD MAC operations, which increases the probability of a fault impacts on the output probabilities. Note that the increase in the precision bitwidth can also reduce the fault criticality since

[1] https://github.com/EEESlab/mobilenet_v1_stm32_cmsis_nn.git
[2] https://www.st.com/en/microcontrollers-microprocessors/stm32h743vi.html

a soft error in a less significant bit is less likely to generate a critical fault. Such behaviour can be seen in Table III *All* when comparing *w8a4* and *w8a8* configurations, where the number of critical faults decreases and tolerable faults increases significantly.

To understand the layers' vulnerability to the soft errors, Table III shows the results obtained from FI campaigns that target distinct layers of the MobileNet CNN topology considering different data volumes. Results show that the most effective faults tend to become either critical or system crashes in low-precision activation configurations (i.e., *a2* and *a4*). Such an effect can be seen in convolutional layers with a high volume of input data processing, where tolerable faults tend to 0% (i.e., layers 1 to 4). In turn, the number of tolerable faults increases as the input data volume reduces in the convolution layers (i.e., layers 25 and 27). This is because the dimensions of input activations are reduced during MobileNet CNN execution, while the channel width increases, thus making the data volume larger in weights w.r.t. input activations. Consequently, faults occurring in these layers are more likely to propagate to the output, i.e., the appearance of a high number of tolerable and critical faults in layers 26 and 27. Note that the occurrence of faults (i.e., Tolerable + Critical + Crash) also increases alongside the precision bitwidth.

C. Applying mitigation techniques to MobileNet CNN

Aiming to reduce the MobileNet CNN susceptibility to soft errors, this Section considers the use of two software-based mitigation techniques: P-TMR and RAT. Both techniques are applied to the matrix multiplication function, which is considered here the most critical one due to its higher active period within the MobileNet execution time.

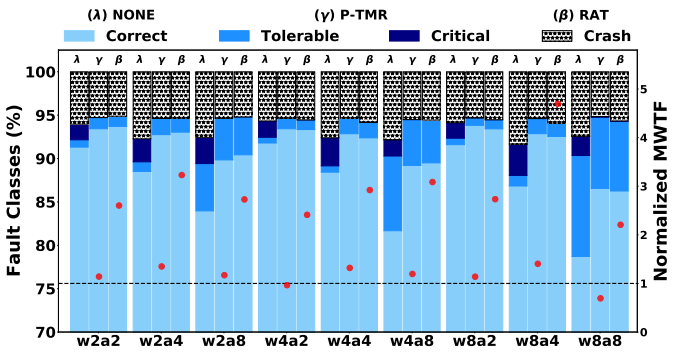
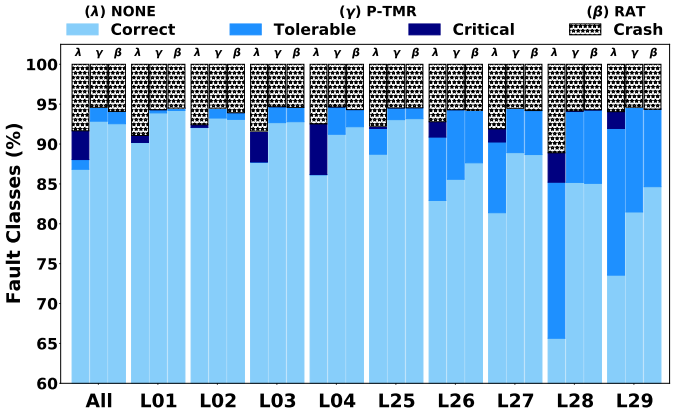


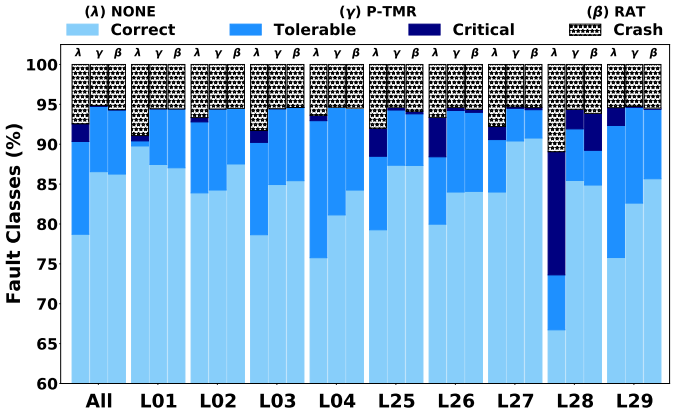
Fig. 3. Results showing fault classifications comparing MobileNet CNN without protection, with P-TMR, and RAT mitigation techniques. The red dots indicate the normalized MWTF (right y-axis).

Figure 3 shows the reliability improvement of MobileNet CNN by applying the two mitigation techniques. The x-axis has three bars for each adopted precision bitwidth configuration. The first bar represents MobileNet CNN with unprotected code (λ), and the other two the mitigation techniques P-TMR (γ) and RAT (β). The left-hand y-axis shows the soft error percentage obtained from the fault injection campaigns, and the right-hand y-axis shows the MWTF normalized by the unprotected version. While the left-hand metric shows an overview of the generated faults, the one at the right-hand side relies on a well-accepted reliability metric to compare the two mitigation techniques.

As expected, Figure 3 shows that both mitigation techniques significantly increase the number of correct outputs while reducing the number of critical faults. In general, a lower precision bitwidth configuration (i.e., 2 and 4-bits to w and a) lead to a reduction in fault occurrences. On the other hand, 8-bit configurations present a higher occurrence of tolerable faults. This is due to larger bitwidth operations that reduce the fault masking rate. Even under these conditions, both mitigation techniques protect the code and turn critical faults into correct or tolerable ones. Figure 3 shows that P-TMR has a significant AVF improvement in all scenarios, but the performance penalty does not compensate for w4a2 and w8a8 configurations (i.e., normalized $MWTF < 1$). In turn, the RAT improved the MWTF in all configurations, raising up to 4.7× in the *w8a4* precision bitwidth.



(a) w8a4



(b) w8a8

Fig. 4. Results of fault classification by layer of the two most affected precision bitwidth configurations.

Figure 4 shows the reliability improvement per layer for the most affected precision bitwidth configurations (i.e., *w8a4* and *w8a8*). Compared to the unprotected version, both mitigation techniques show soft error reliability improvements. On the one hand, Figure 4.a shows a reduction of up to 8% in critical faults and system crashes in the 4-bit precision activation. On the other hand, Figure 4.b illustrates that some tolerable, critical, and crash faults become correct outputs for 8-bit precision activations. In the P-TMR perspective, this

effect occurs mainly due to faults striking registers used by redundant instructions. Unlike, RAT reduces the number of vulnerable registers during the critical function's execution, taking advantage of the inherent high resilience of MobileNet.

D. Trade-off Between Performance and Reliability

This Section details the drawbacks introduced by the two mitigation techniques and discusses the trade-off between increased protection and performance penalty. Figure 5 shows the performance overhead of the P-TMR and RAT compared to no protection execution. The execution times were extracted by running the MobileNet CNN on an STM32H743 board. Results show a performance degradation of up to 1.2× for RAT and 3.8× for P-TMR, depending on the precision bitwidth configuration. In this regard, the original MobileNet achieves ∼1.9 inferences per second. In turn, when applying the P-TMR mitigation technique, the number of inferences per second reduces to ∼0.5 while the RAT ∼1.5 in worst-case scenarios. Note that the most remarkable performance overhead occurs because P-TMR is applied to the application's intermediate code without further optimization, i.e., the application is compiled with *-O2* and the mitigation technique is applied without architecture-specific optimizations. This approach is required to avoid code removals made by the compiler's backend.

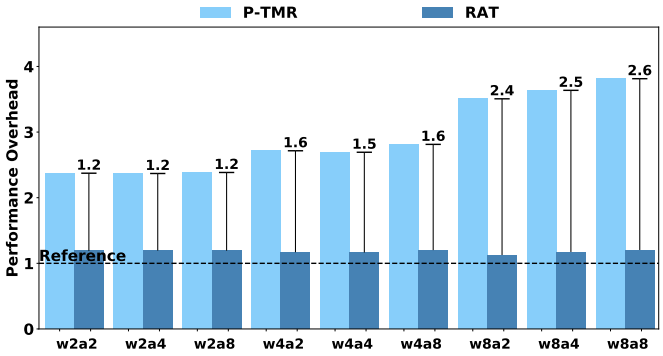


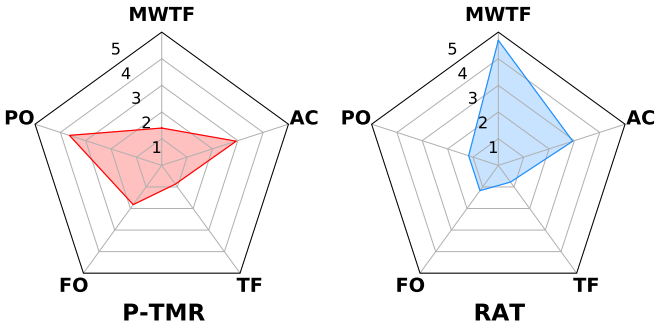
Fig. 5. MobileNet execution time overhead considering P-TMR and RAT.

Figure 6 compares the relative trade-off between reliability, accuracy, performance and memory footprint overhead for two precision bitwidth configurations (*w8a4* and *w8a8*). Table IV shows the footprint overhead, which is calculated based on the additional hardened application code size resulting from both P-TMR and RAT mitigation techniques w.r.t. the original application code (i.e., FLASH memory).

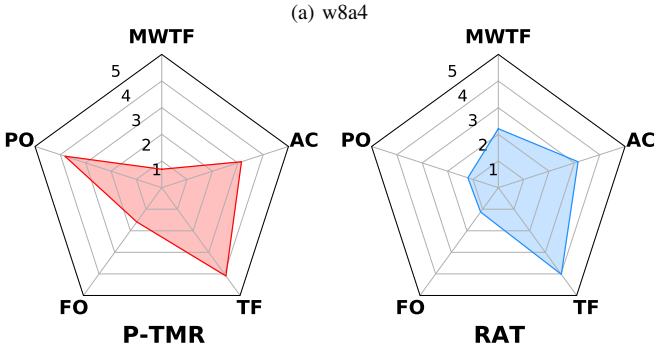
TABLE IV
NORMALIZED MOBILENET FOOTPRINT OVERHEAD WHEN APPLYING SOFT ERROR MITIGATION TECHNIQUES.

	w2a2	w2a4	w2a8	w4a2	w4a4	w4a8	w8a2	w8a4	w8a8
P-TMR	1.78	1.80	1.59	1.80	1.82	1.60	1.81	1.82	1.59
RAT	1.16	1.16	1.13	1.16	1.17	1.13	1.16	1.17	1.13

This comparison provides an overview of the advantages and disadvantages of both mitigation techniques when applied to the MobileNet CNN. Gathered values are normalized between 1 and 5, and the top axis represents the MWTF



(a) w8a4



(b) w8a8

Fig. 6. Relative trade-off between P-TMR and RAT mitigation techniques considering *w8a4* and *w8a8* precision bitwidth configurations, comparing *Mean Work To Failure* (MWTF), *Performance Overhead* (PO), *Footprint Overhead* (FO), *Accuracy* (AC), and *Tolerable Faults* (TF).

improvement, the two left axes represent the performance and memory overheads, and the two right axes show the precision and the tolerable percentages. Figure 6 clearly shows that RAT presents a significant lower performance overhead, which directly led to an improved MWTF w.r.t. the P-TMR.

The resulting performance overhead can be explained not only by the increased number of instructions but also the instruction set employed by each mitigation technique. Table V shows that P-TMR consists of almost 5× more *Thumb* instructions, which correspond to near 90% of the entire hardened code. This highly increase is mainly due to the register spilling forced by the high register pressure and the impact of replicated instructions. In turn, RAT slightly increases the number of executed *Thumb* and *SIMD*, maintaining the percentage of both instruction sets compared to the reference MobileNet execution. Aforementioned results demonstrate that RAT provides the best relative performance, reliability and memory footprint utilisation trade-off.

TABLE V
PERCENTAGE OF USE AND RELATIVE INCREASE IN EXECUTED INSTRUCTIONS CONSIDERING DIFFERENT INSTRUCTION SETS.

	w8a4			w8a8		
Type	**None**	**P-TMR**	**RAT**	**None**	**P-TMR**	**RAT**
SIMD	34.3%	9.8%	32.2%	36.7%	10.1%	30.5%
Thumb	65.7%	90.2%	67.8%	63.3%	89.9%	69.5%
Type	**None**	**P-TMR**	**RAT**	**None**	**P-TMR**	**RAT**
SIMD	1×	1.1×	1.05×	1×	1.1×	1.06×
Thumb	1×	4.8×	1.15×	1×	4.9×	1.23×

These results are of paramount importance for safety-critical applications because, in addition to reliability, these applications have real-time requirements. For example, in self-driving cars, a late reaction can lead to a fatal accident [48]. In this context, traditional soft error mitigation solutions involving time redundancy, such as TMR, may not be suitable for such kind of applications. Even when partially applied, this technique might inflict a significant response time penalty that is not tolerable in real-time applications, thus justifying the need for lightweight techniques, such as RAT, especially for resource constraint systems.

VI. Conclusions

This paper assesses the soft error reliability of the MobileNet CNN when executed on the Arm Cortex-M7 processor architecture. The evaluated results demonstrate that the adopted CNN has a high susceptibility to soft errors in higher precision bitwidth configurations, since the fault occurrence achieves up to 20%. Results also demonstrate that the variation of precision bitwidth of the activations is more susceptible to soft errors than the weights, since critical failures affect both the reliability and the accuracy of CNN. Moreover, the reduction of weights and activations precision bitwidth increases the fault-masking capability of up to 10%, thus reducing the MobileNet CNN susceptibility to the occurrence of soft errors. However, such precision bitwidth reduction does not eliminate the occurrence of critical failures, requiring the use of fault mitigation techniques. Finally, gathered results show that RAT provides significant soft error reliability improvement at a lower performance penalty (i.e., $\sim$1.2$\times$ on average) when compared to the P-TMR.

Future works will focus on two main directions. The first direction comprises the soft error assessment of accelerator-based IoT devices, which are more suitable for highly computing-intensive AI applications. The second investigation aims to assess the MobileNet's behaviour when injecting faults in memory locations, considering different precision bitwidth configurations.

References

[1] V. Joshi, M. L. Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," *Nature Communications*, vol. 11, no. 1, p. 2473, 2020.

[2] J. Amoh and K. M. Odame, "An optimized recurrent unit for ultra-low-power keyword spotting," *ACM on Interactive, Mobile, Wearable andUbiquitous Technologies*, vol. 3, no. 2, 2019.

[3] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for Arm Cortex-M CPUs," *arXiv preprint arXiv:1801.06601*, 2018.

[4] STMicroelectronics, "AI expansion pack for STM32CubeMX," 2020. [Online]. Available: https://www.st.com/en/embedded-software/x-cube-ai.html

[5] A. Capotondi, M. Rusci, M. Fariselli, and L. Benini, "CMix-NN: Mixed low-precision cnn library for memory-constrained edge devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 871–875, 2020.

[6] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine learning for internet of things data analysis:A survey," *Digital Communications and Networks*, vol. 4, no. 3, pp.161–175, 2018.

[7] ISO, "Road vehicles – Functional safety," 2011. [Online]. Available: https://www.iso.org/standard/68383.html

[8] G. Abich, R. Reis, and L. Ost, "The impact of precision bitwidth on the soft error reliability of the MobileNet network," in *IEEE Latin American Symposium on Circuits and Systems (LASCAS)*, 2021, pp. 1–4.

[9] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech, "Selective hardening for neural networks in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp.216–222, 2019.

[10] 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 *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2017.

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[12] J. Gava, R. Reis, and L. Ost, "RAT: A lightweight system-level soft error mitigation technique," in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SOC)*, 2020, pp. 165–170.

[13] A. Avižienis, J.-C. Laprie, and B. Randell, "Dependability and its threats:a taxonomy," in *Building the Information Society*, pp. 91–120, 2004.

[14] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and materials reliability*, vol. 5, no. 3, pp. 305–316, 2005.

[15] B. Nicolescu and R. Velazco, "Detecting soft errors by a purely software approach: method, tools and experimental results," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2003, pp. 57–62.

[16] A. Benso, S. Chiusano, P. Prinetto, and L. Tagliaferri, "A C/C++source-to-source compiler for dependable applications," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2000, pp. 71–78.

[17] A. Serrano-Cases, Y. Morilla, P. Martín-Holgado, S. Cuenca-Asensi, and A. Martínez-Álvarez, "Non-intrusive automatic compiler-guided reliability improvement of embedded applications under proton irradiation," *IEEE Transactions on Nuclear Science*, vol. 66, no. 7, pp. 1500–1509, 2019.

[18] G. S. Rodrigues, F. L. Kastensmidt, R. Reis, F. Rosa, and L. Ost, "Analyzing the impact of using pthreads versus OpenMP under fault injection in ARM Cortex-A9 dual-core," in *European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, 2016, pp. 1–6.

[19] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August, "SWIFT: Software implemented fault tolerance," in *International Symposium on Code Generation and Optimization (CGO)*, 2005, pp. 243–254.

[20] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Error detection by duplicated instructions in super-scalar processors," *IEEE Transactions on Reliability*, vol. 51, no. 1, pp. 63–75, 2002.

[21] M. Didehban and A. Shrivastava, "nZDC: A compiler technique for near zero silent data corruption," in *ACM/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[22] G. A. Reis, J. Chang, and D. I. August, "Automatic instruction-level software-only recovery," *IEEE Micro*, vol. 27, no. 1, pp. 36–47, 2007.

[23] S. Feng, S. Gupta, A. Ansari, and S. Mahlke, "Shoestring: probabilistic soft error reliability on the cheap," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 1, pp. 385–396, 2010.

[24] S. Feng, S. Gupta, A. Ansari, S. A. Mahlke, and D. I. August, "Encore: low-cost, fine-grained transient fault recovery," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011, pp. 398–409.

[25] D. Kuvaiskii, O. Oleksenko, P. Bhatotia, P. Felber, and C. Fetzer, "Elzar: Triple modular redundancy using intel avx (practical experience report)," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2016, pp. 646–653.

[26] M. Didehban, A. Shrivastava, and S. R. D. Lokam, "NEMESIS: A software approach for computing in presence of soft errors," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 297–304.

[27] M. Didehban, S. R. D. Lokam, and A. Shrivastava, "InCheck: An in-application recovery scheme for soft errors," in *ACM/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

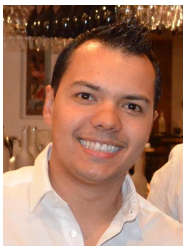
[28] F. Libano, P. Rech, L. Tambara, J. Tonfat, and F. Kastensmidt, "On the reliability of linear regression and pattern recognition feed forward artificial neural networks in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 65, no. 1, pp. 288–295, 2018.

[29] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the

resilience of deep neural networks," in *ACM/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.
[30] 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 Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2019.
[31] F. da Rosa, R. Garibotti, L. Ost, and R. Reis, "Using machine learning techniques to evaluate multicore soft error reliability," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 6, pp. 2151–2164, 2019.
[32] M. G. Trindade, A. Coelho, C. Valadares, R. A. Viera, S. Rey, B. Cheymol, M. Baylac, R. Velazco, and R. P. Bastos, "Assessment of a hardware-implemented machine learning technique under neutron irradiation," *IEEE Transactions on Nuclear Science*, vol. 66, no. 7, pp.1441–1448, 2019.
[33] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardeleben, "Binfi: An efficient fault injector for safety-critical machine learning systems," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2019, pp. 1–23.
[34] M. G. Trindade, R. P. Bastos, R. Garibotti, L. Ost, M. Letiche, and J. Beaucour, "Assessment of machine learning algorithms for near-sensor computing under radiation soft errors," in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2020, pp. 1–4.
[35] L. M. Luza, D. Söderström, G. Tsiligiannis, H. Puchner, C. Cazzaniga, E. Sanchez, A. Bosio, and L. Dilillo, "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)*. IEEE, 2020, pp. 1–6.
[36] G. Abich, J. Gava, R. Reis, and L. Ost, "Soft error reliability assessment of neural networks on resource-constrained IoT devices," in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2020, pp. 1–4.
[37] S. Kundu, K. Basu, M. Sadi, T. Titirsha, S. Song, A. Das, and U. Guin, "Reliability analysis for ML/AI hardware," *arXiv preprint arXiv:2103.12166*, 2021.
[38] V. Bandeira, F. Rosa, R. Reis, and L. Ost, "Non-intrusive fault injection techniques for efficient soft error vulnerability analysis," in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2019, pp. 123–128.
[39] Imperas Software, "Open Virtual Platforms (OVP)," 2021. [Online]. Available: http://www.ovpworld.org/
[40] G. Abich, R. Garibotti, V. Bandeira, F. da Rosa, J. Gava, F. Bortolon, G. Medeiros, F. G. Moraes, R. Reis, and L. Ost, "Evaluation of the soft error assessment consistency of a JIT-based virtual platform simulator," *IET Computers & Digital Techniques*, vol. 15, no. 2, pp. 125–142, 2021.
[41] N. Khoshavi, C. Broyles, and Y. Bi, "A survey on impact of transient faults on BNN inference accelerators," *arXiv preprint arXiv:2004.05915*, 2020.
[42] C. Lattner and V. Adve, "LLVM: a compilation framework for lifelong program analysis transformation," in *International Symposium on Code Generation and Optimization (CGO)*, 2004, pp. 75–86.
[43] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, D. I. August, and S. S. Mukherjee, "Software-controlled fault tolerance," *ACM Transactions on Architecture and Code Optimization*, vol. 2, no. 4, pp. 366–396, 2005.
[44] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*. IEEE, 2003, pp. 29–40.
[45] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.
[46] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers," *arXiv preprint arXiv:1905.13082*, 2019.
[47] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2009, pp. 502–506.
[48] S. S. Banerjee, S. Jha, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp.586–597.

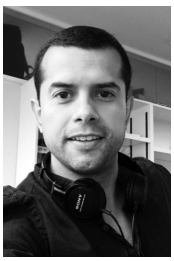
Geancarlo Abich received a bachelor's degree in computer engineering from the University of Santa Cruz do Sul (UNISC) in 2014. In 2015, he started his MSc in Computer Science at Federal University of Rio Grande do Sul (UFRGS), and followed his Ph.D. in 2017 at the same institution. For the past six years, he has been researching and developing tools involving the implementation and evaluation of reliable embedded systems based on resource constrained devices. His research activity focuses on modeling and simulation of robust MPSoCs and deep learning approaches targeting resource constrained devices.

Jonas Gava received a bachelor's degree in computer engineering from the Federal University of Rio Grande do Sul (UFRGS) in 2019. In 2020 started as an MSc Student in Microelectronics and followed his PhD in 2021 at the same institution. For the past three years, he has been researching and developing tools involving the implementation and evaluation of software-based soft error mitigation techniques. He is an IEEE student member.



Rafael Garibotti (M'14) is an Associate Professor at PUCRS University. Former Visiting Scholar at Université Grenoble Alpes, France. Former Postdoctoral Fellow at both the prestigious School of Engineering and Applied Sciences of Harvard University and UFRGS, Brazil. He received his Ph.D. and MSc. Degree in Microelectronics, respectively from the University of Montpellier and EMSE, France and his BSc. Degree in Computer Engineering from PUCRS University, Brazil. His research activity focuses on AI safety, robotics and autonomous systems, multicore architectures, hardware accelerator and robust deep learning.

Ricardo Reis (M'81–SM'06) received the Electrical Engineering degree from the Federal University of Rio Grande do Sul (UFRGS), Brazil, in 1978, and the Ph.D. degree in informatics, option microelectronics from the Institut National Polytechnique de Grenoble, France, in 1983. He received the Doctor Honoris Causa from University of Montpellier, France, in 2016. He has been a Full Professor with UFRGS since 1981. He is at research level 1A of the CNPq (Brazilian National Science Foundation), and the head of several research projects supported by government agencies and industry. He has published over 700 papers in journals and conference proceedings and authored or co-authored several books. His current research interests include physical design, physical design automation, design methodologies, digital design, EDA, circuits tolerant to radiation, and microelectronics education. Prof. Reis was a recipient of the IEEE Circuits and Systems Society (CASS) Meritorious Service Award 2015. He was the Vice President of the IEEE CASS representing Region 9 (Latin America) and president of the Brazilian Computer Society (SBC).



Luciano Ost is currently a Faculty Member with Loughborough University's Wolfson School - UK. He received his Ph.D. degree in Computer Science from PUCRS, Brazil in 2010. During his Ph.D., Dr Ost worked as an invited researcher at the Microelectronic Systems Institute of the Technische Universitaet Darmstadt (from 2007 to 2008) and at the University of York (October 2009). After the completion of his doctorate, he worked as a research assistant (2 years) and then as an assistant professor (2 years) at the University of Montpellier II in France. He has authored more than 90 papers and his research is devoted to advancing hardware and software architectures to improve performance, security, and reliability of life-critical and multiprocessing embedded systems.