Assessment of Machine Learning Algorithms for Near-Sensor Computing under Radiation Soft Errors

M. Garay Trindade, R. Garibotti, L. Ost, M. Letiche, J. Beaucour, and R. Possamai Bastos

***Abstract*—Machine learning (ML) algorithms have been regaining momentum thanks to their ability to analyze substantial and complex data, supporting artificial intelligence decisions in cloud computing but also in near-sensor computing in endpoint devices. Both cloud and near-sensor computing are liable to radiation-induced soft errors, especially in automotive and aerospace safety-critical applications. In this regard, this paper contributes by comparing the accuracy of two prominent machine learning algorithms running on a low-power processor upset by radiation-induced soft errors. Both ML algorithms have been assessed with the help of a fault injection-based method able to natively emulate soft errors directly in a development board. In addition, neutron radiation test results suggest the most critical situations in which mitigation solutions should address.**

***Index Terms*—Radiation-induced soft errors, machine learning algorithms, low-power processors.**

I. Introduction

Machine learning (ML) algorithms have evolved rapidly as an effective solution for a broad range of applications in different industry segments, ranging from cloud computing to the Internet of Things. Aiming to boost the use of underlying algorithms, researchers and industry leaders are investing heavily in the exploration of more efficient instruction set architectures [1], ML inference processors [2], and accelerators [3]. While ML benefit from the substantial amount of resources (e.g., processors, memory) of large-scale computing systems, executing these computationally intensive algorithms under resource-constrained devices such as near-sensor computing endpoint devices is still a challenging task. To overcome the constraint above, researchers are investigating software libraries/APIs aiming to support the use of efficient ML algorithms at reduced memory footprint, which is critical to resource-constrained devices [4]. Another approach relies on the development of lightweight and optimized ML algorithms, allowing their execution on low-power processors typically embedded in near-sensor computing endpoint devices. These seemingly benefits come at the cost of precision, which is also of particular importance in assessing overall ML algorithms accuracy and applicability.

ML algorithms are employed to recognize patterns and predict how systems (e.g., medical, automotive) would react to unexpected circumstances. The use of the convolutional neural network (CNN)-based pedestrian detection in cars is a prominent example of ML algorithms application. For such systems, software engineers must develop not only lightweight and performance efficient ML algorithms, but also more secure and reliable algorithms aiming to guarantee a fail-safe system operation.

With that in mind, researchers have started to investigate the impact of radiation-induced soft errors on the reliability of ML techniques. For instance, the authors in [5] [6] examine effects of soft errors in CNNs. In [7] [8], different CNN implementations have been analyzed using an FPGA-based fault injection approach, which emulates the occurrence of faults by modifying the bitstream configuration. In turn, Santos et al. [9] investigate how the presence of soft errors in GPUs can reduce the reliability of a CNN. Rosa et al. [10] investigate the impact of soft error on an automotive vehicle application that is based on CNN. Results show that the occurrence of soft errors affects the vehicle travel. Authors in [11] have proposed a framework that performs fault injection at specific deep neural networks (DNN) design points across the weights, activations, and hidden states. Results show that the resilience varies across DNNs depending on the model and data type. While [12] has conducted soft error resilience analysis of Bayesian machines, the work in [13] has focused on a binary support vector machine implemented in an FPGA.

Different from the above works, this paper assesses and compares the reliability of two ML algorithms – feed-forward artificial neural network (ANN) and support vector machine (SVM) – running on a popular low-power processor (Arm Cortex-M4) under effects of radiation-induced soft errors. Gathered results have been obtained through neutron radiation tests conducted with a neutron generator as well as an emulation-based fault injection campaign to better understand how radiation-induced soft errors affect the reliability of the case-study ML algorithms.

II. Case-Study ML Algorithm Models

This section briefly describes two prominent ML algorithm models that have been studied in this work: ANN and SVM. Both models are commonly used in classification tasks, which consist of previously learning underlying behaviors of a set of known data through a training phase, allowing a computing system (at a later time) classifying new data observations (herein input vectors) accordingly.

This work has been partially supported by: IRT Nanoelec (ANR-10-AIRT-05) and LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01), both funded by the French government program "Investissement d'avenir"; and MultiRad project funded by Région Auvergne-Rhône-Alpes's international ambition pack.

M. Garay Trindade and R. Possamai Bastos are with Univ. Grenoble Alpes, CNRS, Grenoble INP*, TIMA, 38000 Grenoble, France. (e-mail: firstname.lastname@univ-grenoble-alpes.fr)

R. Garibotti is with the School of Technology, Pontifical Catholic University of Rio Grande do Sul, Porto Alegre 90619-900, Brazil.

L. Ost is with Loughborough University, Loughborough LE11 3TU, U.K.

M. Letiche and J. Beaucour are with the Institut Laue-Langevin, 38042 Grenoble, France.

*Institute of Engineering Univ. Grenoble Alpes

1) ANN Model: The basic components of an ANN are the neurons that are organized in a layered structure. The first layer forwards the input vector to be classified to the first neurons. A neuron receives as input the outputs of the neurons of the previous, performing a weighted addition of them, evaluating the result on an activation function, and forwarding the outcome to the next layer. Each neuron on the last layer represents one of the possible previously-defined classes, and the one with the greatest value determines which class has been finally identified by the ANN. The weights are calculated beforehand during a training phase.

2) SVM Model: The principle of a binary SVM is modeling – during the training phase – a linear classifier the best separates two previously-defined classes, enabling hence identifying at a later time the class of an input vector. This linear classifier is represented as a weighted sum of the input vectors used in the training phase. In situations demanding more than two classes (multiclass classification), a binary SVM can be trained for classifying between each pair of classes, choosing the class the most commonly assigned by binary SVMs. This approach is known as One-vs-One.

III. Fault Injection-Based Assessment Method

This section describes the method for assessing the reliability of the case-study ML algorithm running under effects of radiation-induced soft errors. This is based on campaigns of single fault injections (single soft errors) that are emulated during the execution of the case-study program (herein an ML algorithm) – running natively on the target computing system under test (SUT) – and remotely configured via the popular software debugger GDB from a control computer. The SUT is composed of a target low-power processor, data memory, program memory, and on-board peripheral devices able to communicate with an external control computer.

The method assesses the classification reliability of a ML algorithm under the influence of single soft errors by counting how often it classifies an input vector correctly, i.e. if the ML algorithm properly identifies the previously-defined class of the input vector. Furthermore, the method also assesses the ML algorithm's inability to tolerate soft errors provoking either *Computing Crashes* or *Critical Failures*. The method workflow comprises five phases (Figure 1): (1) generation of golden reference results; (2) specification of fault injection profiles; (3) fault injection campaign; (4) assessment of results; and (5) statistical analysis of results.

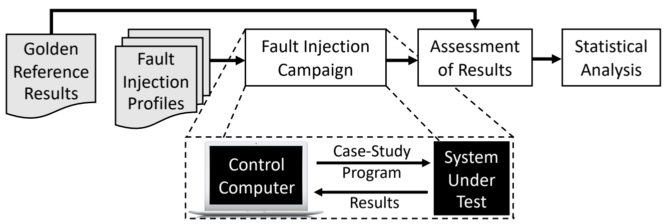

Figure 1. Fault injection-based method for assessing the reliability of an ML algorithm (case-study program) running on a low-power processor (SUT subcircuit) under effects of single soft errors.

1) Generation of Golden Reference Results: The case-study ML algorithm is natively executed in the SUT under fault-free circumstances (no presence of faults) in order to generate its golden reference results.

2) Specification of Fault Injection Profiles: In order to compare the reliability of different case-study ML algorithms under possible scenarios of single soft errors in volatile memory elements of the processor, each case study is assessed under the same set of fault injection profiles, which is defined according to the pseudo-code in Algorithm 1.

From the population of possible input vectors, a small sample is randomly selected. The same criterion is applied to the population of fault injection instants at which a memory bit is inverted for modeling a single soft error. Each fault injection instant is simplified here as a discrete time unit represented by the execution period of each instruction of the case-study program.

For the sake of separately analyzing the single soft error impact on the processor memory bits, the criterion of assessment and comparison is set to exclusively cover all memory bits of the processor registers, considering thus the hypothesis that the data memory, program memory, and other on-board peripheral devices are protected by soft error mitigation techniques.

Algorithm 1 Set of Fault Injection Profiles

```
1: for x in [small sample of input vectors] do
2:     for y in [small sample of fault injection instants] do
3:         for z in [set of processor registers] do
4:             for w in [set of processor register bits] do
5:                 FaultInjectionProfile(x, y, z, w)
```

3) Fault Injection Campaign: Each fault injection profile is remotely emulated in the SUT through the command "set" in the software debugger GDB executed in the control computer. The case-study ML algorithm is thus run several times on the SUT according to Algorithm 1, each run emulating a different fault injection profile and providing a result from the computation of a given input vector by the ML algorithm.

4) Assessment of Results: Results are compared against their golden reference according to the following situations:

– *No Failure*: The result of the ML algorithm does not differ from a golden reference, i.e. the ML algorithm correctly computes an input vector and provides a fault-free output vector;
– *Tolerable Failure*: There is a mismatch between the result of the ML algorithm and the golden reference; however, the resulting class is correct, i.e. the sign of the result of the ML algorithm and the golden reference are the same;
– *Computing Crash*: The processor suddenly stops operating, providing no valid results from the ML algorithm but a useful indication that radiation effects have upset the SUT and it must be restarted and compute again the input vector;
– *Critical Failure*: There is a mismatch between the result of the ML algorithm and the golden reference, and the resulting class is also not correct.

5) Statistical Analysis of Results: Equation 1 below analyzes the ratio of the total number of *Critical Failures* to the total number of fault injection profiles assessed according to

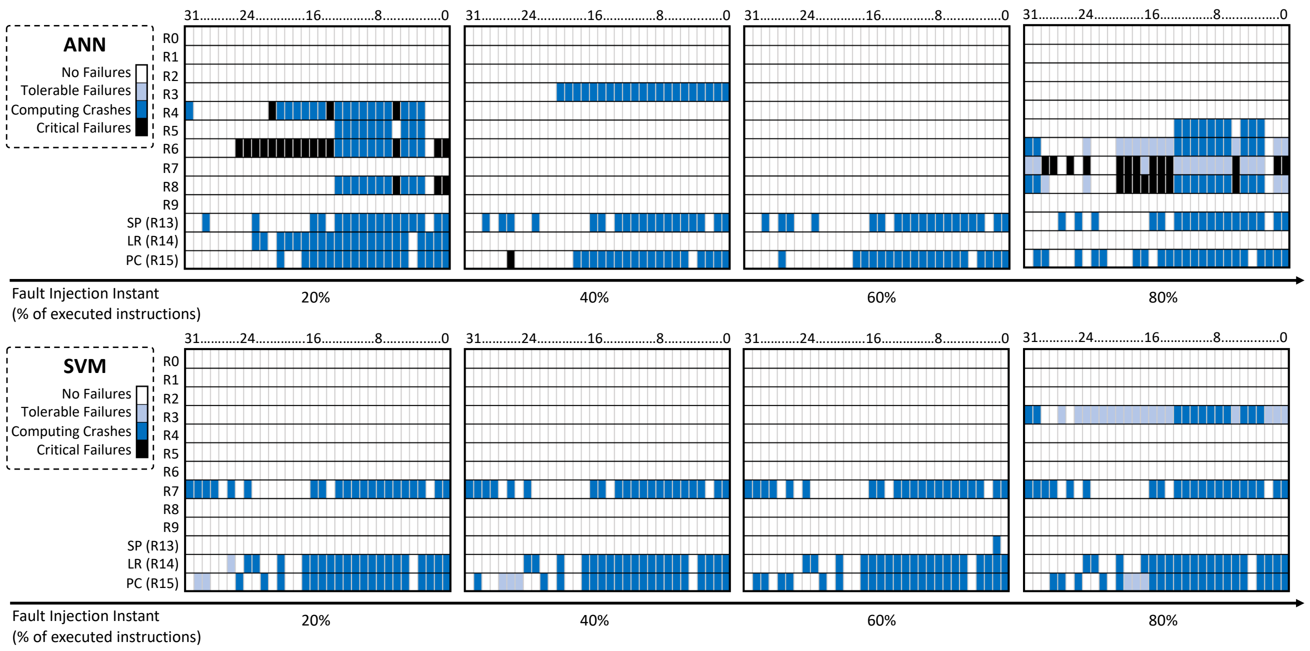

Figure 2. Snapshots of the fault injection instants in the processor register map for ANN (top) and SVM (bottom) algorithms using the same set of input vectors. This shows which registers are further stimulated and therefore the most susceptible to radiation-induced soft errors.

the specification in Algorithm 1. Similar equations compile also the ratios of *Computing Crashes*, *Tolerable Failures*, and *No Failures*. As the described method assesses the same scenarios of single soft errors as well as the same dataset, these equations allow thus comparing different ML algorithms, suggesting the most reliable solutions for correctly classifying input vectors of a given dataset even though the processor is upset by single soft errors.

$$\%CriticalFailures = \frac{\#CriticalFailures \times 100}{\#FaultInjectionProfiles} \quad (1)$$

The fault injection campaign in accordance with Algorithm 1 requires the emulation of a large number of fault injection profiles that would make the assessment impractical if small samples are not taken from the populations of possible input vectors, fault injection instants, processor registers, and processor register bits [14]. Hence, a small sample of input vectors, a small sample of fault injection instants, and the entire populations of processor registers and processor register bits are combined through the aforementioned equations, being considered each one a small sample of a normally distributed population. The traditional Student's t-distribution based on such samples of small size is thus applied to estimate the means of these populations.

IV. Fault Injection-Based Assessment

This section analyzes results of experiments that have been carried out applying the method described in section III for comparing the case-study ML algorithms defined in section II.

1) Description of Experiments: The STM32 NUCLEO-L45RE-P development kit has been used as the SUT, which includes the low-power processor Arm Cortex-M4. While the SVM algorithm has been implemented in C language, the ANN algorithm has been optimized using the STM32 X-CUBE-AI package, which is a software that generates program code from a high-level description of an ANN. The Iris flower dataset [15] has been used to train both case-study ML algorithms before the fault injection campaign. The dataset consists of samples representing flowers from three different species. The fault injection campaign experiment has indeed assessed ML algorithms already trained, while operating for classifying samples (herein input vectors) of a dataset. The program code of the case-study ML algorithms have been loaded one at a time into the SUT using GDB. Furthermore, a custom-built script automates the implementation of the pseudo-code in Algorithm 1 for both case-study ML algorithms.

For the specification of fault injection profiles according to section III, one sample (input vector) has been randomly taken from each flower specie to maintain the diversity of the original dataset, making thus the small sample of input vectors defined in Algorithm 1. On the other hand, the small sample of fault injection instants has been taken considering the following criterion: firstly injecting a single soft error when 20% of the instructions of the case-study program (ANN or SVM) have been executed. After in a new run of the case-study ML algorithm, when 40% of the instructions have been executed, and so on for 40%, 60%, and 80%. Regarding the set of processor registers, only the ones used by the case-study ML algorithms have been assessed, covering all their bits. The only register that the method is not able to assess is the *$CONTROL* as a fault injected causes the GDB to disconnect from the SUT.

2) Analysis of Results by Register: Figure 2 shows effects of single soft errors in processor registers when running the case-study ML algorithms, both classifying the same set of input vectors from the Iris flower dataset. The revealed situations are illustrated by general-purpose registers (from *R0* to *R9*) and control registers (*SP*, *LR*, and *PC*).

For general-purpose registers, most faults have produced *No Failure*. These registers are normally used to manipulate data and implement calculations. In this sense, if the program has a large number of variables or intensive calculations, it is possible to frequently back up its values in the system

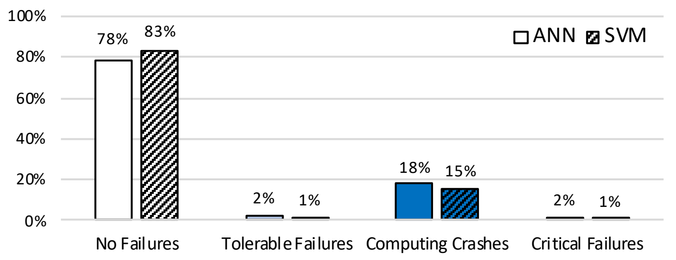

Figure 3. Summary of situations induced by the fault injection campaign in the ANN and SVM algorithms.

stack, and depending on the moment when an error occurs, a register can no longer be used or be naturally rewritten by the code, which explains most non-occurrence of failures. When a injected fault manifests a failure (including crashes), Figure 2 shows that a *Computing Crash* is more likely to occur. As general-purpose registers are used to store pointers to tables that contain the weights used by the ML algorithm, an error in them may cause the program to fail whenever it tries to use it to access the weights. However, the most dangerous is when a register bit flips and it is storing an intermediate calculation value from the ML algorithm. This error can spread silently to the final values and affect (*Critical Failures*) or not (*Tolerable Failures*) the final classification issued.

On the other hand, control registers are always very sensitive to faults, as they are directly linked to memory. For example, the *PC* register contains the address of the next instruction to be executed by the processor. If one of its bits flips, the code may jump to an invalid memory location or interrupt the execution flow. In regard to *LR*, the failures are more likely to happen due to the different levels of nested function calls. In the SVM, there are no nested function calls, while there are in the ANN. When a nested function call takes place, a common practice is to store the *LR* value, which contains the return address of the current function being executed, in the system stack at the beginning of a function and reload the value before returning. Therefore, our SVM implementation is more sensitive to faults on the *LR* w.r.t the ANN, as shown in Figure 2. In contrast, *SP* is more sensitive to ANN than SVM, as our implementation makes heavy use of the stack.

3) Analysis of Global Results: Figure 3 presents the summary of the fault injection campaigns for the ANN and SVM algorithms. The values have been calculated as shown in section III-5, combining all results.

Looking closely in Figure 3, *Critical Failures*, although crucial, are very rare, falling below 2% for both ML algorithms. This shows the robustness of the output presented by the two ML algorithms when they do not stop by *Computing crashes*.

In general, Figure 3 suggests that the SVM algorithm presents a slightly better reliability than the ANN when under the influence of single soft errors.

V. Radiation Test-Based Assessment

The case-study ML algorithm ANN has been tested under a neutron generator in the same SUT used in the fault injection campaign. The experiment has been conducted at the TOMOH9 beam line located at the Institut Laue-Langevin (ILL). The TOMOH9 is a tomography beam line with a spectrum of neutron energy between 1 to 2 MeV and a flux during the experiment on the order of $1.8 \cdot 10^5$ neutrons / $(cm^2 \cdot s)$.

The SUT has been irradiated for 4 hours and 30 minutes, yielding a total fluence of $2.916 \cdot 10^9$ neutrons / cm^2. During this period, 24 faults have been detected, accounting for a cross-section of $8.230 \cdot 10^{-9}$ cm^2. Among those, 3 faults have provoked *Critical Failures*, 19 faults have led the SUT to *Computing Crashes*, and 2 faults have manifested as *Tolerable Failures*. Note that the high number of *Computing Crashes* in the radiation test corroborates the results obtained in the fault injection campaign, which shows the *Computing Crashes* are the most common situations, suggesting that possible solutions of soft error mitigation should preferentially focus on addressing them.

VI. Conclusions and Ongoing Works

This paper provides findings suggesting the case-study ML algorithm SVM is slightly more reliable than the ANN to classify the same data observations under scenarios of single soft errors in the processor. In addition, neutron radiation tests of the ANN show that the majority of detected faults produces *Computing Crashes*, being them naturally detectable without any additional fault detection technique but requiring to compute again the ANN operation.

In terms of future works, we intend to perform a deeper comparison of both ML through additional fault injection campaigns. We are currently broadening the coverage of our fault injection campaign to obtain more details of the effect of faults on our implementations.

References

[1] Y. Chen, et al. An instruction set architecture for machine learning. *ACM Transactions on Computer Systems*, 36(3):1–35, August 2019.
[2] Arm ethos-n77 processor - datasheet. 2020.
[3] A. Reuther, et al. Survey and benchmarking of machine learning accelerators. In *IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9, September 2019.
[4] L. Lai, et al. CMSIS-NN: efficient neural network kernels for arm cortex-m cpus. January 2018.
[5] G. Li, et al. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, November 2017.
[6] M. A. Neggaz, et al. Are CNNs reliable enough for critical applications? an exploratory study. *IEEE Design & Test*, 37(2):76–83, 2020.
[7] F. Libano, et al. On the reliability of linear regression and pattern recognition feedforward artificial neural networks in fpgas. *IEEE Transactions on Nuclear Science*, 65(1):288–295, January 2018.
[8] F. Libano, et al. Selective hardening for neural networks in fpgas. *IEEE Transactions on Nuclear Science*, 66(1):216–222, January 2019.
[9] F. F. d. Santos, et al. Analyzing and increasing the reliability of convolutional neural networks on gpus. *IEEE Transactions on Reliability*, 68(2):663–677, June 2019.
[10] F. R. da Rosa, et al. Using machine learning techniques to evaluate multicore soft error reliability. *IEEE Transactions on Circuits and Systems–I: Regular papers*, 66(6):2151–2164, June 2019.
[11] B. Reagen, et al. Ares: A framework for quantifying the resilience of deep neural networks. In *Design Automation Conference (DAC)*, pages 1–6, June 2018.
[12] A. Coelho, et al. On the robustness of stochastic bayesian machines. *IEEE Transactions on Nuclear Science*, 64(8):2276–2283, August 2017.
[13] M. G. Trindade, et al. Assessment of a hardware-implemented machine learning technique under neutron irradiation. *IEEE Transactions on Nuclear Science*, 66(7):1441–1448, July 2019.
[14] R. Possamai Bastos, et al. Comparing transient-fault effects on synchronous and on asynchronous circuits. In *IEEE International On-Line Testing Symposium (IOLTS)*, pages 29–34, June 2009.
[15] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, September 1936.