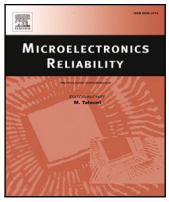
Microelectronics Reliability 150 (2023) 115092

Contents lists available at ScienceDirect

Microelectronics Reliability

journal homepage: www.elsevier.com/locate/microrel



SMART: Selective MAC zero-optimization for neural network reliability under radiation

Anuj Justus Rajappa [a,*], Philippe Reiter [a], Tarso Kraemer Sarzi Sartori [b,c], Luiz Henrique Laurini [b], Hassen Fourati [c], Siegfried Mercelis [a], Jeroen Famaey [a], Rodrigo Possamai Bastos [b]

[a] IDLab, University of Antwerp - imec, Sint-Pietersvliet 7, 2000 Antwerpen, Belgium
[b] Univ. Grenoble Alpes, CNRS, Grenoble INP, TIMA, 38000 Grenoble, France
[c] Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, GIPSA-Lab, 38000 Grenoble, France

ARTICLE INFO

Dataset link: https://doi.org/10.5281/zenodo.7986836

Keywords:
Sparsity
Radiation
Optimization
Neural network
Machine learning
Reliability

ABSTRACT

Neural networks running on low-power edge devices can help in achieving ubiquitous computing with limited infrastructure. When such edge devices are deployed in conventional and extreme environments without the necessary shielding, they must be fault tolerant for reliable operation. As a pilot study, we focused on embedding fault tolerance into neural networks by proposing a novel selective multiply-accumulate zero-optimization technique based on whether the value of an input provided to a neuron of a neural network is zero. If the value is zero, then the corresponding multiply-accumulate operation is bypassed. We subjected the operating system-based implementation of the optimization technique to radiation test campaigns using approximately 14 MeV neutrons, and found the proposed optimization technique to improve the fault tolerance of the tested neural network by approximately 44%.

1. Introduction

Machine learning (ML) algorithms for making decisions at the edge [1] and reducing the data transferred between edge devices can alleviate the strain on networks and cloud infrastructure [2]. Thus, when targeting ubiquitous computing [3], machine learning algorithms can allow increasing the quantity of the raw data processed and edge devices deployed even when limited by cloud infrastructure. Edge devices are typically placed close to the data source [2], which could expose them to cosmic rays, hazardous radiation levels, extreme temperatures, unreliable power supplies, etc. [4] at ground level [5], in space, and within nuclear facilities and other hard to reach environments [6]. This exposure can cause transient errors [7], typically manifested as single bit-flips in the edge devices, with the potential to cause system failures [8]. Such failures in mission-critical applications, such as civil and military aviation [9], medical devices [10,11], autonomous vehicles [8,12], UAVs [13], aerospace vehicles [14–16], and nuclear power plants [17–19], can lead to critical consequences [8,20]. Hence, these edge systems must be fault tolerant for reliable operation, which is usually achieved using a combination of hardware [7,21] and software techniques [22].

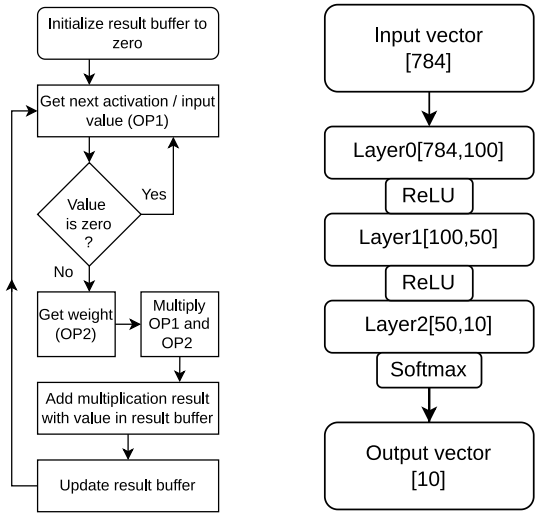
We hypothesized that the fault tolerance of a neural network (NN) can be increased by reducing the number of data transfers and overall execution time. The latter can be achieved by replacing longer-executing Multiply ACcumulate (MAC) operations with shorter-executing zero comparisons, while the former involves reducing the number of arithmetic floating point operations (FLOPs).

The number of FLOPs was reduced by leveraging the sparsity (ratio between the number of non-significant values and the total number of values) of the runtime input values [23] through all the layers of a NN. If an input value to be multiplied with a network weight is zero, then the corresponding MAC operation, which consists of FLOPs, is bypassed. We termed this optimization Selective Multiply-Accumulate zeRo-opTimization (SMART). A process flow diagram for SMART is shown in Fig. 1(a).

The rationale behind the MAC bypass is that zero multiplied by any real number is zero, and zero is also the additive identity for real numbers. Hence, when an addition or multiplication is carried out between two operands, the results can be directly deduced from the operands if at least one operand is zero, without using an adder or multiplier [24,25]. The number of zero comparators replacing FLOPs is proportional to the input sparsity.

* Corresponding author.
E-mail addresses: anuj.justusrajappa@uantwerpen.be (A.J. Rajappa), philippe.reiter@uantwerpen.be (P. Reiter), tarso.kraemer-sarzi-sartori@univ-grenoble-alpes.fr (T.K.S. Sartori), luiz-henrique.laurini@univ-grenoble-alpes.fr (L.H. Laurini), hassen.fourati@grenoble-inp.fr (H. Fourati), siegfried.mercelis@uantwerpen.be (S. Mercelis), jeroen.famaey@uantwerpen.be (J. Famaey), rodrigo.bastos@univ-grenoble-alpes.fr (R.P. Bastos).

https://doi.org/10.1016/j.microrel.2023.115092
Received 2 June 2023; Accepted 3 July 2023
Available online 1 October 2023



(a) Overview of SMART. (b) Architecture of the NN.

Fig. 1. Overview of the SMART and NN architecture.

SMART can be implemented through software changes. We consider SMART to be novel as we could not identify a similar technique for NN fault tolerance among the current state-of-the-art techniques for the operating system (OS) based computing platforms. The closest we could find was the exploration of the relationship between static sparsity of weights and fault resiliency of NNs [26]. While SMART can be achieved in hardware [24,25], it would require specialized processor architectures, unlike our proposed software-based approach that can be executed on commercial off-the-shelf processors.

2. Related works

A number of techniques, such as N-Module Redundancy (NMR) [27,28], Error Detection And Correction (EDAC) codes [29–31], Built-In Current Sensor (BICS) [32,33], bit interleaving with single error correction (SEC) codes [34], Algorithm-Based Fault Tolerance (ABFT) and Result Checking (RC) [35,36] exists to make general purpose computing more reliable. While these techniques can be adopted to improve the NN fault tolerance against soft errors, SMART does not fit under any of the above techniques as it does not involve error detection or correction. Rather, SMART focuses on error prevention by reducing the probability of a soft error occurring in NNs.

For instance, the NMR technique uses multiple identical modules for computation. The output from those computations is majority voted to get the final output. That way, should an error occur in one of those computed outputs, it can be detected and masked through voting. Depending on the number of modules used, this technique has other names such as Dual Module Redundancy (DMR) [37–39], Triple Module Redundancy (TMR) [40–43], and 5MR. Depending on the mode of computation, the NMR technique can be temporal (multiple computations separated in time), spatial (multiple computations separated in space, such as with multiple hardware) or a combination of both. However, NMR has a significant resource consumption overhead.

EDAC uses redundant bits for handling errors in the memory. BICS monitors the SRAM power bus for detecting single-event upset-induced abnormalities. The power bus checking is performed on the SRAM columns and it is combined with a single-parity bit per RAM word to perform error correction. Bit interleaving uses the idea of spacing the bits belonging to a word at an optimal interleaving distance to reduce the chances of multiple bit flips affecting a single word. The combination of bit interleaving and SEC codes can be used to protect the memories from multiple upsets.

ABFT uses the knowledge of the algorithm to inject additional calculations into the algorithm whose outcome is already known. Any deviation in this known outcome signals that an error has occurred. However, RC verifies the results of various mathematical functions without the knowledge of the algorithm used to compute those functions. One advantage that some ABFT algorithms have over RC is in terms of fault localization and error correction [35].

Apart from general techniques, several other optimization techniques for improving the fault tolerance of NNs have been proposed [44]. One proposal uses the Feature-map and Inference Level Resilience (FILR) [45] technique for statically protecting vulnerable parts of a Convolutional Neural Network (CNN) by duplicating the corresponding logical operations and rerunning vulnerable inferences by analyzing their output. Another uses model compression techniques, such as binary quantization, for improving the fault tolerance of a Deep Neural Network (DNN) [26]. Ranger [8] is another technique used to improve the fault tolerance of a DNN by correcting transient faults without re-computation. Compiler-based techniques such as Register Allocation Technique (RAT) explicitly restrict the registers used by certain functions to reduce the exposed area, thereby reducing the CNN's susceptibility to soft errors [46,47].

Others have evaluated the effects of neutron radiation and simulated fault injections on machine learning algorithms like Support Vector Machines (SVMs) [48,49] and CNNs [50], and assessed the fault tolerance of these algorithms.

Studies linking the reliability of CNNs on FPGAs to their parameters and metrics, such as model accuracy, degree of parallelism, quantization and reduced data precision, [16,51] have also been conducted.

The effect of instruction set architecture on the reliability of CNNs has also been studied [52] on an ARM platform with simulated fault injections. However, the study uses the Common Microcontroller Software Interface Standard-NN (CMSIS-NN) [53] library for CNN execution with low-precision fixed-point representation and does not consider runtime input sparsity.

The following sections describe the NNs subjected to the radiation test campaigns; the effects of SMART and temporal TMR techniques on the NNs; the test setup and methodology; analysis of the radiation test results; and, concluding observations and future work.

Table 1
Sparsity of input values to different layers in the NN.

	Layer 0	Layer 1	Layer 2
Sparsity	80.7%	66.6%	47.3%

3. Case study algorithms

The architecture of the NN used during the radiation test campaign is shown in Fig. 1(b). This NN was designed, trained and evaluated using the TensorFlow [54] Python library, and 60 000 training images and 10 000 testing images from the Modified National Institute of Standards and Technology (MNIST) database. This NN is also known as an MNIST digit classifier as the NN is used to classify the images representing digits from 0 to 9. The input sparsity to the different layers of the NN generated at runtime is shown in Table 1, which was computed using all 10 000 test images from MNIST.

The parameters of the trained network are fed to a custom implementation of the NN algorithm in C language, using a custom framework to create four different versions of the NN. These are: (1) a version of the NN without any of the proposed optimization (**Simple**), (2) NN with SMART (**SMART**), (3) an NN with TMR (**TMR**), and (4) an NN with TMR and SMART (**TMR+SMART**).

Table 2
Analysis of results from February and July test campaigns.

NN version	Avg. Neutron flux (10^5 neutrons/cm^2)/s	Irradiation time (h)	Iterations	Number of errors			Neutron fluence (10^{10} neutrons/cm^2)	Cross section (10^{-10} cm^2)
				Tolerable	Critical	Total		
Simple	4.27	12.0	1892	11	0	11	1.85	5.94
SMART	3.84 4.27	43.7 56.3	452 1195	29 17	2 1	49	14.7	3.33
TMR	4.27	11.9	1813	5	0	5	1.83	2.73
TMR+SMART	3.84 4.27	44.5 56.0	451 1181	23 13	0 3	39	14.8	2.64

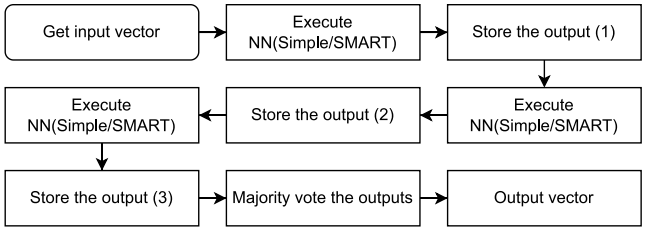


Fig. 2. Flowchart of the TMR versions of NN.

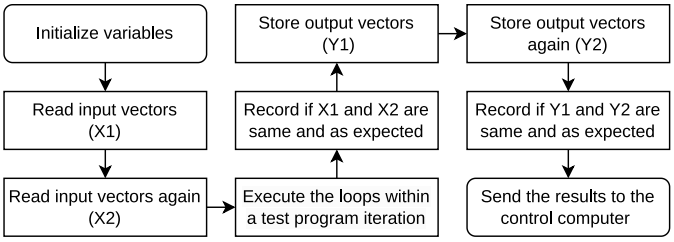


Fig. 3. Flow chart of the radiation test program.

3.1. NN(Simple) version

The Simple NN version of the case study algorithm contains the implementation for the NN inferencing algorithm without any of the proposed optimization. It is expected to provide a reference for lower fault tolerance compared to the NN(SMART) version while executing on OS-based systems.

3.2. NN(SMART) version

The SMART NN version of the case study algorithm contains the implementation for the NN inferencing algorithm with the proposed SMART technique. It is expected to provide higher fault tolerance to NN inferencing algorithms compared to the NN(Simple) version while executing on OS-based systems.

3.3. NN(TMR) version

The TMR NN version of the case study algorithm contains the implementation for the NN inferencing algorithm protected with the temporal TMR technique, where the entire NN(Simple) implementation is executed thrice successively and the majority output is voted, as shown in Fig. 2. This implementation is similar to the one by Czajkowski and colleagues [43]. However, all three instances of the NN(Simple) implementation are executed without conditionally skipping any of them. This version is expected to provide a reference for higher fault tolerance than the NN(SMART) version for comparison purposes.

3.4. NN(TMR+SMART) version

The TMR+SMART NN version is similar to the NN(TMR) version. However, instead of executing the NN(Simple) implementation, NN(SMART) implementation is executed thrice and the output of the NN(TMR+SMART) version is the majority-voted output of those three executions, as shown in Fig. 2. This version is expected to provide higher fault tolerance to the NN algorithms compared to all the other versions in the case study algorithm as both the TMR and SMART techniques implemented in this hybrid version are expected to work constructively in protecting the NN algorithms from radiation-induced soft errors.

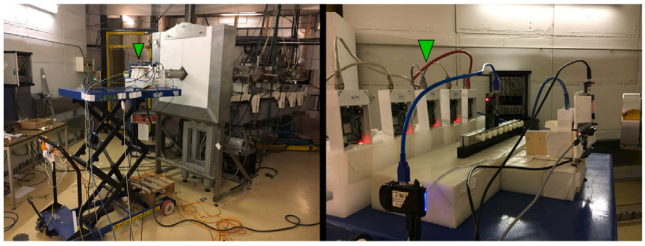


Fig. 4. Radiation test setup inside the radiation test chamber with the system under test (SUT).

4. Radiation test setup

Each of the four versions of the NN algorithm was packaged into separate radiation test programs, whose flowchart is shown in Fig. 3, to facilitate executing the case study algorithms on the radiation test setup developed by Université Grenoble Alpes (UGA) [55]. The test setup contains the system under test, which is exposed to neutron radiation during the experiments as shown in Fig. 4, and the control computer, which is placed outside the radiation test chamber to control and communicate with the system under test.

The number of iterations of the test program is controlled by the radiation test setup and each iteration corresponds to an execution of the test program. To limit the size of a test program, 250 inputs were randomly selected from the MNIST testing images, and inference results for all of these images were computed in one iteration. To reduce the variables in the experiments, a single input data set was used across all campaigns. Each of the inputs contains a one-dimensional array of size 784 in single-precision floating-point format (FP32), which is obtained by normalizing and flattening the two-dimensional array of 28 × 28 pixels representing the resolution of an image in the MNIST database. Each input is used to compute 120 inferences within one iteration. This number was chosen to cause the total execution time of one iteration to lie between 10 s to 20 s, for optimal scheduling of the test programs during the radiation test campaigns. The various loops within an iteration of the test program are shown in Fig. 5. Each NN inference generates a one-dimensional array of size 10 in FP32 as output, where each element in the output corresponds to the probability of the input being an image of a digit from 0 to 9.

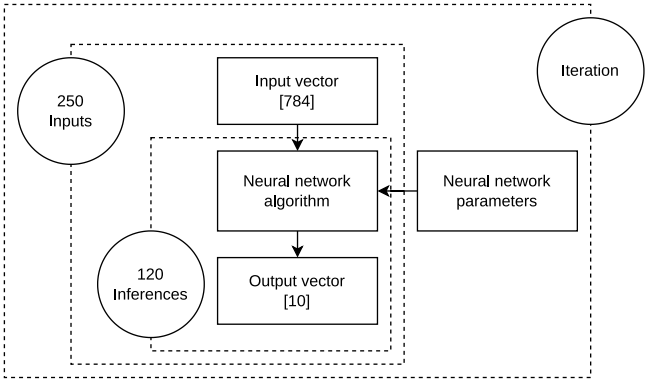


Fig. 5. Loops within an iteration of the radiation test program.

5. Experiment and analysis

The radiation test programs were executed from February 17–18 and July 4–8 of 2022 at Laboratory of Subatomic Physics & Cosmology (LPSC) in Grenoble, France. The radiation test setup utilizes two Raspberry Pi 4B, with Raspberry Pi OS Lite version 11 and a superscalar quadcore Cortex-A72 (ARM v8) 64-bit SoC, where one acts as the system under test and another as the control computer. Once the results were obtained from the experiments, the error analysis was done by comparing the results with the golden reference, which was obtained by running the radiation test program under normal operating conditions.

The analysis of the results from the radiation test campaigns is shown in Table 2 and the corresponding dataset is publicly available [56]. The first column of this table represents the four versions of the NN algorithm tested under radiation. The second column contains the corresponding average neutron flux to which the various NN versions were exposed. The third column represents the time spent by each NN version, executing under various neutron flux levels, on the setup's multi-core CPU. The fourth column represents the number of iterations of the radiation test program for each NN version under various neutron flux levels. The fifth column represents the number of errors that occurred during all the corresponding iterations of the NN versions. This column is divided into three sub-columns which represent the following error counts:

1. *Tolerable error* is incremented by one if single or multiple errors occurred within an iteration but did not result in any classification mismatch when compared with the golden reference.
2. *Critical error* is incremented by one if single or multiple errors occurred within an iteration and includes classification mismatches when compared with the golden reference.
3. *Total error* is the sum of tolerable and critical error counts for all NN versions. This error count is used for cross section [57] calculation.

The sixth column represents the neutron fluence associated with each NN version. This represents the total number of neutrons that passed through a 1 cm^2 area of the radiation setup while the corresponding NN version was executing. The last column represents the cross section calculated from the total error count for each of the NN versions. Note that the data explicitly pertaining to any program crashes or hangs experienced by the system under test during the experiments are currently not available.

By design, a new set of result files is sent by the test program as output, along with its duplicate for each iteration. For each NN version, the results and their duplicates were analyzed separately. This results in two values for metrics such as iterations and number of errors, which should be equal under normal conditions. However, under radiation, if an error occurred outside the case study algorithm (such as during the output write process), these values could be different. Hence, as a means to determine the commonality between these two sets of results, the minimum count values between them are further analyzed, as shown in Table 2.

Table 3
Errors associated with input and output of radiation test program.

NN version	Skipped files	Input error	Output error
Simple	5	1	1
SMART	39	0	17
TMR	7	2	2
TMR+SMART	40	0	15

6. Observation and discussion

The last column of Table 2 suggests that SMART has reduced the cross section – i.e., the probability of either a tolerable or critical error occurring for a given neutron radiation –, thus, increasing the overall fault tolerance of NN(SMART) compared to NN(Simple). NN(TMR) has superior fault tolerance compared to NN(SMART) but at an increased computational cost. Finally, NN(TMR+SMART) outperforms NN(TMR) by a small margin in terms of fault tolerance improvement. However, more radiation tests are required to confirm this advantage as the observed margin is small. Future radiation tests will ensure approximately equal irradiation time for all tested algorithms.

The Record blocks shown in the flowchart of the test program (c.f., Fig. 3) are responsible for detecting an error in the input or output of the case study algorithms, regardless of an error within the execution of the case study algorithms, using DMR technique [37,39]. Sometimes the result files, including the duplicates, sent by the test program running on top of the OS in the system under test were found to be either empty or, corrupted and unreadable, rendering them useless for further analysis. Hence, those files were skipped from being subjected to analysis. These skipped file counts along with the input and output error count for each case study algorithm are shown in Table 3. Table 3 suggests that SMART versions experienced a higher number of file skips compared to non-SMART versions during analysis. This is because the SMART versions were irradiated for prolonged periods of time compared to non-SMART versions, as evident from Table 2.

Table 4 shows various characteristics of the radiation test program such as the execution time, memory consumption and failure in time (FIT) [58]. The FIT was calculated for each of the test programs by multiplying the approximate sea level neutron flux value of 0.13 $cm^{-2}\ h^{-1}$ with 10^9 hours and cross section values shown in Table 2. The FIT values are therefore proportional to the cross section values. The approximate sea level neutron flux value of 0.13 $cm^{-2}\ h^{-1}$ for 14 MeV neutrons, assuming 1 MeV energy bin/width, was calculated by integrating the reference neutron differential flux function (c.f., Equation A.1 in [58]) between the 13.5 MeV lower limit and 14.5 MeV upper limit, and multiplying the integration result by 3600 s h^{-1}.

The size of the executable was measured using the GNU size utility [59] with System V conventions. As evident from the size column of Table 4, only the .text section which contains the program code [60] seems to be varying due to changes in the instructions associated with various case study algorithms, while the rest of the executable remains the same size, which is as expected by design.

The execution time for the radiation test programs was calculated using time stamps immediately obtained before and after the execution of the test program.

The Send result block at the end of the test program flow (c.f., Fig. 3) was disabled to mitigate the variable effects of the standard output process during the execution time measurement. The execution time measurement was carried out 100 times for each test program containing a case study algorithm and the statistical average of those measurements is shown in the execution time column of Table 4.

Table 4
Characteristics of the radiation test program.

NN version	Execution time (s)	Size (bytes) .text	Size (bytes) Total	FIT
Simple	4.59	4216	1 130 066	0.077
SMART	17.89	4228	1 130 078	0.043
TMR	4.60	4720	1 130 570	0.036
TMR+SMART	17.88	4732	1 130 582	0.034

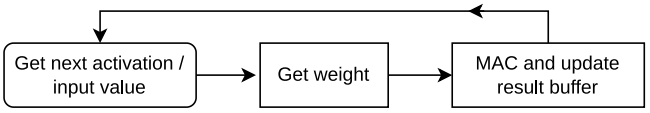


Fig. 6. Overview of MAC operation implementation in non-SMART versions.

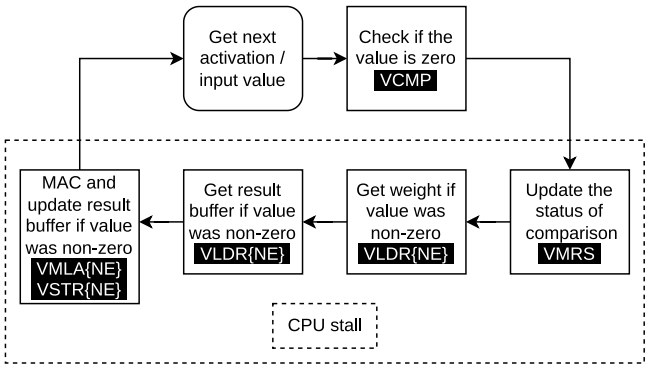


Fig. 7. Overview of MAC operation implementation in SMART versions [61].

One thing to note is that the radiation test program associated with both the corresponding TMR and non-TMR versions takes almost the same amount of time to execute, despite the redundancy. This is because the total number of inferences was always kept at 120 (c.f., Fig. 5 and Section 4), i.e., while 120 inferences were carried out for non-TMR versions for each input, only 40 inferences were carried out for each input in the TMR versions, with each TMR inference requiring 3 corresponding non-TMR inferences.

6.1. Spatial and temporal footprints

Another observation from Table 4 is that the SMART versions have a 12-byte memory overhead and about 4 times the execution time overhead compared to the corresponding non-SMART versions. Further analysis into the differences brought to the NN algorithms by the SMART at the assembly level revealed the following:

- VMLA (floating-point multiply-accumulate) [61] instructions associated with MAC operations on the input values of the NN algorithms were executed in the non-SMART versions unconditionally and does not block the CPU. An overview of the MAC operation implementation in the non-SMART versions is shown in Fig. 6.
- VMLA instructions associated with MAC operations on the input values of the NN algorithms were executed in the SMART versions conditionally. This conditional execution implementation adds five more instructions per execution compared to the corresponding non-SMART versions, where four of them are VFP (floating point) instructions [61,62]. These additional instructions include VCMP (floating-point compare) followed by VMRS (transfer the contents of a VFP system register to an ARM register), which stalls the processor until all the current VFP instructions such as VLDR (extension register load), VMLA and VSTR (extension register store) are executed [61]. An overview of the conditional implementation is diagrammed in Fig. 7.

The above points explain the reason behind the significant increase in the execution time for SMART versions. However, according to Tables 2 and 4, despite the overhead disadvantage, the cross section and FIT for SMART are lower compared to non-SMART versions. This suggests that even though both SMART and non-SMART versions use and execute VFP instructions, executing the VMLA instruction in particular (which carries out the MAC operation), significantly reduces the fault tolerance of the NN algorithms. Thus, lowering the number of VMLA instruction executions (which is the case with SMART versions) increases the fault tolerance of these algorithms. One plausible reason behind the relationship between the VMLA instruction and the fault tolerance against neutron-induced soft errors could be the overall spatial footprint of the data using this instruction, i.e., the chip area the data needs to travel increases with VMLA instruction, giving more neutrons a chance to alter this data. If that is the case, then despite the temporal footprint increase caused by the SMART (prolonged exposure to the neutron beam, c.f., Table 4), the positive effect of the overall spatial footprint decrease by SMART on NN fault tolerance is higher. This is plausible since the cross section is a function of area (neutron flux) and irradiation time. Thus, the chip area and execution time used by the algorithms plays a critical role in the number of soft errors faced by these algorithms. However, more beam time and targeted tests are required to confirm and improve the statistical significance of this observation.

Supporting this observation, a similar relationship between the susceptibility to soft error and the associated (spatial) memory footprint was proposed by Geancarlo Abich and colleagues [12] for CNN's implemented with SIMD (single instruction, multiple data) instructions. They established that the variation in spatial and temporal footprint alters the NN's susceptibility to soft errors and "while boosting the performance, SIMD instructions supported by Arm Cortex-M4 and M7 processors increase the memory footprint and the CNN susceptibility to soft errors" [12].

6.2. Error analysis granularity

As evident from Section 5, the error counts were incremented, at maximum, by one per iteration of the radiation test program. This was deliberate, even though the granularity of the error counts could be theoretically increased to track the errors in the loops within each iteration (c.f., Fig. 5). During the analysis of the results, we observed that increasing the granularity also increased the complexity of the analysis, as the files containing the results were sometimes corrupted. These file corruptions also suggest that soft errors impacted the operations of the OS running on the system under test.

Since the test program is executed from the start for each iteration, by design, a new set of result files is created for each iteration (c.f., Section 5). The unreadable corrupted result files were easily identifiable and skipped (c.f., Table 3). The readable result files, regardless of the corruption, were processed as mentioned in Section 5 (especially the last paragraph) to mitigate the effects of corruption. Assuming the probability of multiple soft errors occurring within one iteration to be negligible, we selected the error count granularity to be at the iteration level.

7. Conclusion and future work

We proposed a novel optimization technique called SMART to improve the fault tolerance of NN inferencing algorithms running on OS-based computing systems.

As described in Section 3, to understand the impact of SMART, radiation test campaigns for evaluating the fault tolerance of the four NN versions, namely NN(Simple), NN(SMART), NN(TMR) and NN(TMR+SMART) were conducted. Analysis of the results (c.f., Table 2 and Section 6) from these campaigns suggests that NN(SMART) outperformed NN(Simple) by approximately 44% in terms of the reduction in the probability of a neutron-induced error occurring (fault tolerance). However, the TMR NNs exhibited higher fault tolerance compared to the non-TMR NNs, as expected. For instance, NN(TMR) had approximately 18% fault tolerance improvement over NN(SMART), but NN(TMR+SMART) marginally performed better than NN(TMR) by approximately 3.3% – whose statistical significance can be improved with more experiments.

For future work, we will evaluate the performance of SMART in a bare-metal system. Furthermore, the overhead of SMART could be significantly reduced by replacing FP32 comparison with integer comparison. The relationship between input sparsity, different NN output functions [23], as well as different NN architectures and SMART will also be explored. To improve error analysis granularity, an alternative methodology also needs to be devised for observing and storing the results from the radiation test programs.

CRediT authorship contribution statement

Anuj Justus Rajappa: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Philippe Reiter:** Conceptualization, Investigation, Writing – review & editing, Resources, Supervision, Project administration, Funding acquisition. **Tarso Kraemer Sarzi Sartori:** Investigation, Software, Writing – review & editing. **Luiz Henrique Laurini:** Investigation, Software, Writing – review & editing. **Hassen Fourati:** Funding acquisition, Software, Resources. **Siegfried Mercelis:** Supervision. **Jeroen Famaey:** Supervision. **Rodrigo Possamai Bastos:** Conceptualization, Investigation, Writing – review & editing, Resources, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Dataset link: https://doi.org/10.5281/zenodo.7986836.

Acknowledgments

This work was supported in part by : the MultiRad project of the Région Auvergne-Rhône-Alpes's international ambition pack (PAI), France; the French national research agency, France within the France-2030 program (ANR-15-IDEX-0002); the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01); the IRT Nanoelec (ANR-10-AIRT-0005) of the French national program PIA ("Programme d'Investissements d'Avenir); the LPSC's GENESIS platform; Bourse de mobilité Génération IA 2030, funded by the French embassy in Belgium; and MOVIQ (Mastering Onboard Vision Intelligence and Quality) project funded by Flanders Innovation & Entrepreneurship (VLAIO), Belgium and Flanders Space (VRI), Belgium.

References

[1] R. Kukunuri, A. Aglawe, J. Chauhan, K. Bhagtani, R. Patil, S. Walia, N. Batra, Edgenilm: Towards NILM on edge devices, in: Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, BuildSys '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 90–99, http://dx.doi.org/10.1145/3408308.3427977.

[2] W. Shi, S. Dustdar, The promise of edge computing, Computer 49 (5) (2016) 78–81, http://dx.doi.org/10.1109/MC.2016.145.

[3] J. Krumm, Ubiquitous Computing Fundamentals, CRC Press, 2018, p. 2, URL https://www.routledge.com/Ubiquitous-Computing-Fundamentals/Krumm/p/book/9781420093605.

[4] M.A. Hanif, F. Khalid, R.V.W. Putra, S. Rehman, M. Shafique, Robust machine learning systems: Reliability and security for deep neural networks, in: 2018 IEEE 24th International Symposium on on-Line Testing and Robust System Design (IOLTS), 2018, pp. 257–260, http://dx.doi.org/10.1109/IOLTS.2018.8474192.

[5] T. O'Gorman, The effect of cosmic rays on the soft error rate of a DRAM at ground level, IEEE Trans. Electron Devices 41 (4) (1994) 553–557, http://dx.doi.org/10.1109/16.278509.

[6] J. Prinzie, F.M. Simanjuntak, P. Leroux, T. Prodromakis, Low-power electronic technologies for harsh radiation environments, Nat. Electr. 4 (4) (2021) 243–253, http://dx.doi.org/10.1038/s41928-021-00562-4.

[7] S. Sayil, Single event soft error mechanisms, in: Soft Error Mechanisms, Modeling and Mitigation, Springer International Publishing, Cham, 2016, pp. 31–48, http://dx.doi.org/10.1007/978-3-319-30607-0_4.

[8] Z. Chen, G. Li, K. Pattabiraman, A low-cost fault corrector for deep neural networks through range restriction, in: 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2021, pp. 1–13, http://dx.doi.org/10.1109/DSN48987.2021.00018.

[9] E. Kulida, V. Lebedev, About the use of artificial intelligence methods in aviation, in: 2020 13th International Conference "Management of Large-Scale System Development" (MLSD), 2020, pp. 1–5, http://dx.doi.org/10.1109/MLSD49919.2020.9247822.

[10] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, K. Huang, Toward an intelligent edge: Wireless communication meets machine learning, IEEE Commun. Mag. 58 (1) (2020) 19–25, http://dx.doi.org/10.1109/MCOM.001.1900103.

[11] Amisha, P. Malik, M. Pathania, V. Rathaur, Overview of artificial intelligence in medicine, J. Fam. Med. Prim. Care 8 (7) (2019) 2328, http://dx.doi.org/10.4103/jfmpc.jfmpc_440_19.

[12] G. Abich, R. Garibotti, R. Reis, L. Ost, The impact of soft errors in memory units of edge devices executing convolutional neural networks, IEEE Trans. Circuits Syst. II 69 (3) (2022) 679–683, http://dx.doi.org/10.1109/TCSII.2022.3141243.

[13] P. McEnroe, S. Wang, M. Liyanage, A survey on the convergence of edge computing and AI for UAVs: Opportunities and challenges, IEEE Internet Things J. 9 (17) (2022) 15435–15459, http://dx.doi.org/10.1109/JIOT.2022.3176400.

[14] G. Furano, et al., Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities, IEEE Aerosp. Electron. Syst. Mag. 35 (12) (2020) 44–56, http://dx.doi.org/10.1109/MAES.2020.3008468.

[15] S. Chien, R. Doyle, A. Davies, A. Jonsson, R. Lorenz, The future of AI in space, IEEE Intell. Syst. 21 (4) (2006) 64–69, http://dx.doi.org/10.1109/MIS.2006.79.

[16] F. Libano, P. Rech, B. Neuman, J. Leavitt, M. Wirthlin, J. Brunhaver, How reduced data precision and degree of parallelism impact the reliability of convolutional neural networks on FPGAs, IEEE Trans. Nucl. Sci. 68 (5) (2021) 865–872, http://dx.doi.org/10.1109/TNS.2021.3050707.

[17] S.K.M. Balgehshiri, A.Z. Paydar, B. Zohuri, Artificial intelligence driven nuclear power reactors (A technical memorandum), J. Energy Power Eng. 16 (2022) 71–80, http://dx.doi.org/10.17265/1934-8975/2022.02.003.

[18] K. Zhang, C. Hutson, J. Knighton, G. Herrmann, T. Scott, Radiation tolerance testing methodology of robotic manipulator prior to nuclear waste handling, Front. Robot. AI 7 (2020) http://dx.doi.org/10.3389/frobt.2020.00006.

[19] J. Chen, J. Klein, Y. Wu, S. Xing, R. Flammang, M. Heibel, L. Zuo, A thermoelectric energy harvesting system for powering wireless sensors in nuclear power plants, IEEE Trans. Nucl. Sci. 63 (5) (2016) 2738–2746, http://dx.doi.org/10.1109/TNS.2016.2606090.

[20] A. Vega, P. Bose, A. Buyuktosunoglu, Chapter 1 - introduction and chapter 2 - reliable and power-aware architectures: Fundamentals and modeling, in: A. Vega, P. Bose, A. Buyuktosunoglu (Eds.), Rugged Embedded Systems, Morgan Kaufmann, Boston, 2017, pp. 1–37, http://dx.doi.org/10.1016/B978-0-12-802459-1.00001-4.

[21] JSC, Space radiation effects on electronic components in low-earth orbit, 1999, https://llis.nasa.gov/lesson/824, [Online; accessed 04-March-2023].

[22] Q. Huang, J. Jiang, An overview of radiation effects on electronic devices under severe accident conditions in NPPs, rad-hardened design techniques and simulation tools, Prog. Nucl. Energy 114 (2019) 105–120, http://dx.doi.org/10.1016/j.pnucene.2019.02.008.

[23] A. Apicella, F. Donnarumma, F. Isgrò, R. Prevete, A survey on modern trainable activation functions, Neural Netw. 138 (2021) 14–32, http://dx.doi.org/10.1016/j.neunet.2021.01.026.

[24] M. Masadeh, O. Hasan, S. Tahar, Input-conscious approximate multiply-accumulate (MAC) unit for energy-efficiency, IEEE Access 7 (2019) 147129–147142, http://dx.doi.org/10.1109/ACCESS.2019.2946513.

[25] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S.W. Keckler, W.J. Dally, SCNN: An accelerator for compressed-sparse convolutional neural networks, SIGARCH Comput. Archit. News 45 (2) (2017) 27–40, http://dx.doi.org/10.1145/3140659.3080254.

[26] M. Sabbagh, C. Gongye, Y. Fei, Y. Wang, Evaluating fault resiliency of compressed deep neural networks, in: 2019 IEEE International Conference on Embedded Software and Systems (ICESS), 2019, pp. 1–7, http://dx.doi.org/10.1109/ICESS.2019.8782505.

[27] P. Balasubramanian, D. Maskell, N. Mastorakis, Majority and minority voted redundancy scheme for safety-critical applications with error/no-error signaling logic, Electronics 7 (11) (2018) http://dx.doi.org/10.3390/electronics7110272.

[28] Koren, Su, Reliability analysis of N-modular redundancy systems with intermittent and permanent faults, IEEE Trans. Comput. C-28 (7) (1979) 514–520, http://dx.doi.org/10.1109/TC.1979.1675397.

[29] R.W. Hamming, Error detecting and error correcting codes, Bell Syst. Tech. J. 29 (2) (1950) 147–160, http://dx.doi.org/10.1002/j.1538-7305.1950.tb00463.x.

[30] G.-C. Yang, Reliability of semiconductor RAMs with soft-error scrubbing techniques, IEE Proc. - Comput. Digit. Tech. 142 (1995) 337–344, (7), URL https://digital-library.theiet.org/content/journals/10.1049/ip-cdt_19952162.

[31] M.Y. Hsiao, A class of optimal minimum odd-weight-column SEC-DED codes, IBM J. Res. Dev. 14 (4) (1970) 395–401, http://dx.doi.org/10.1147/rd.144.0395.

[32] R. Possamai Bastos, J.-M. Dutertre, M. Garay Trindade, R.A.C. Viera, O. Potin, M. Letiche, B. Cheymol, J. Beaucour, Assessment of on-chip current sensor for detection of thermal-neutron-induced transients, IEEE Trans. Nucl. Sci. 67 (7) (2020) 1404–1411, http://dx.doi.org/10.1109/TNS.2020.2975923.

[33] F. Vargas, M. Nicolaidis, SEU-tolerant SRAM design based on current monitoring, in: Proceedings of IEEE 24th International Symposium on Fault- Tolerant Computing, 1994, pp. 106–115, http://dx.doi.org/10.1109/FTCS.1994.315652.

[34] P. Reviriego, J.A. Maestro, S. Baeg, S. Wen, R. Wong, Protection of memories suffering MCUs through the selection of the optimal interleaving distance, IEEE Trans. Nucl. Sci. 57 (4) (2010) 2124–2128, http://dx.doi.org/10.1109/TNS.2010.2042818.

[35] P. Prata, J. Silva, Algorithm based fault tolerance versus result-checking for matrix computations, in: Digest of Papers. Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (Cat. No.99CB36352), 1999, pp. 4–11, http://dx.doi.org/10.1109/FTCS.1999.781028.

[36] R. Granat, K.L. Wagstaff, B. Bornstein, B. Tang, M. Turmon, Simulating and detecting radiation-induced errors for onboard machine learning, in: 2009 Third IEEE International Conference on Space Mission Challenges for Information Technology, 2009, pp. 125–131, http://dx.doi.org/10.1109/SMC-IT.2009.22.

[37] A. Mahmoud, S.K.S. Hari, C.W. Fletcher, S.V. Adve, C. Sakr, N. Shanbhag, P. Molchanov, M.B. Sullivan, T. Tsai, S.W. Keckler, Hardnn: Feature map vulnerability evaluation in CNNs, 2020, http://dx.doi.org/10.48550/arXiv.2002.09786, arXiv:2002.09786.

[38] CNET, Meet tesla's self-driving car computer and its two AI brains, 2023, https://www.cnet.com/tech/computing/meet-tesla-self-driving-car-computer-and-its-two-ai-brains/, (Online; accessed on 17 Apr 2023).

[39] B.K. Kim, Reliability analysis of real-time controllers with dual-modular temporal redundancy, in: Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No. PR00306), IEEE, 1999, pp. 364–371.

[40] J. Wakerly, Microcomputer reliability improvement using triple-modular redundancy, Proc. IEEE 64 (6) (1976) 889–895, http://dx.doi.org/10.1109/PROC.1976.10239.

[41] R.E. Lyons, W. Vanderkulk, The use of triple-modular redundancy to improve computer reliability, IBM J. Res. Dev. 6 (2) (1962) 200–209, http://dx.doi.org/10.1147/rd.62.0200.

[42] S. Esposito, S. Avramenko, M. Violante, On the consolidation of mixed criticalities applications on multicore architectures, in: 2016 17th Latin-American Test Symposium (LATS), 2016, pp. 57–62, http://dx.doi.org/10.1109/LATW.2016.7483340.

[43] D. Czajkowski, M. Pagey, P. Samudrala, M. Goksel, M. Viehman, Low power, high-speed radiation hardened computer & flight experiment, in: 2005 IEEE Aerospace Conference, 2005, pp. 1–10, http://dx.doi.org/10.1109/AERO.2005.1559559.

[44] F. Su, C. Liu, H.-G. Stratigopoulos, Testability and dependability of AI hardware: Survey, trends, challenges, and perspectives, IEEE Des. Test 40 (2) (2023) 8–58, http://dx.doi.org/10.1109/MDAT.2023.3241116.

[45] A. Mahmoud, S.K. Sastry Hari, C.W. Fletcher, S.V. Adve, C. Sakr, N. Shanbhag, P. Molchanov, M.B. Sullivan, T. Tsai, S.W. Keckler, Optimizing selective protection for CNN resilience, in: 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE), 2021, pp. 127–138, http://dx.doi.org/10.1109/ISSRE52982.2021.00025.

[46] G. Abich, J. Gava, R. Garibotti, R. Reis, L. Ost, Applying lightweight soft error mitigation techniques to embedded mixed precision deep neural networks, IEEE Trans. Circuits Syst. I. Regul. Pap. 68 (11) (2021) 4772–4782, http://dx.doi.org/10.1109/TCSI.2021.3097981.

[47] J. Gava, et al., A lightweight mitigation technique for resource-constrained devices executing DNN inference models under neutron radiation, IEEE Trans. Nucl. Sci. (2023) 1, http://dx.doi.org/10.1109/TNS.2023.3262448.

[48] M. Garay Trindade, F. Benevenuti, M. Letiche, J. Beaucour, F. Kastensmidt, R. Possamai Bastos, Effects of thermal neutron radiation on a hardware-implemented machine learning algorithm, Microelectron. Reliab. 116 (2021) 114022, http://dx.doi.org/10.1016/j.microrel.2020.114022.

[49] M.G. Trindade, A. Coelho, C. Valadares, R.A.C. Viera, S. Rey, B. Cheymol, M. Baylac, R. Velazco, R.P. Bastos, Assessment of a hardware-implemented machine learning technique under neutron irradiation, IEEE Trans. Nucl. Sci. 66 (7) (2019) 1441–1448, http://dx.doi.org/10.1109/TNS.2019.2920747.

[50] H.-B. Wang, Y.-S. Wang, J.-H. Xiao, S.-L. Wang, T.-J. Liang, Impact of single-event upsets on convolutional neural networks in xilinx zynq FPGAs, IEEE Trans. Nucl. Sci. 68 (4) (2021) 394–401, http://dx.doi.org/10.1109/TNS.2021.3062014.

[51] F. Libano, B. Wilson, M. Wirthlin, P. Rech, J. Brunhaver, Understanding the impact of quantization, accuracy, and radiation on the reliability of convolutional neural networks on FPGAs, IEEE Trans. Nucl. Sci. 67 (7) (2020) 1478–1484, http://dx.doi.org/10.1109/TNS.2020.2983662.

[52] G. Abich, J. Gava, R. Reis, L. Ost, Soft error reliability assessment of neural networks on resource-constrained IoT devices, in: 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2020, pp. 1–4, http://dx.doi.org/10.1109/ICECS49266.2020.9294951.

[53] L. Lai, N. Suda, V. Chandra, Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus, 2018, arXiv preprint arXiv:1801.06601.

[54] M. Abadi, et al., TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, Software available from tensorflow.org, URL https://www.tensorflow.org/.

[55] T. Kraemer Sarzi Sartori, H. Fourati, M. Letiche, R.P. Bastos, Assessment of radiation effects on attitude estimation processing for autonomous things, IEEE Trans. Nucl. Sci. 69 (7) (2022) 1610–1617, http://dx.doi.org/10.1109/TNS.2022.3176676.

[56] A. Justus, T.K.S. Sartori, L.H. Laurini, R.P. Bastos, Experimental dataset: SMART for reliable NN inference in a neutron-irradiated OS-based system, 2023, http://dx.doi.org/10.5281/zenodo.7986836.

[57] F. Wrobel, Y. Aguiar, C. Marques, G. Lerner, R. García Alía, F. Saigné, J. Boch, An analytical approach to calculate soft error rate induced by atmospheric neutrons, Electronics 12 (1) (2023) http://dx.doi.org/10.3390/electronics12010104.

[58] JEDEC, Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices, Std., JESD89B, 2021, https://www.jedec.org/system/files/docs/JESD89B.pdf, [Online; accessed 21-May-2023].

[59] GNU, 6. size, https://ftp.gnu.org/old-gnu/Manuals/binutils-2.12/html_chapter/binutils_6.html, [Online; accessed 21-May-2023].

[60] Texas Instruments, 7.3. Introduction to sections, 2023, https://software-dl.ti.com/codegen/docs/tiarmclang/compiler_tools_user_guide/compiler_manual/intro_to_object_modules/introduction-to-sections-stdz0691509.html, (Online; accessed on 30 Mar 2023).

[61] ARM, ARM compiler armasm user guide version 5.06, VFP instructions, 2023, https://developer.arm.com/documentation/dui0473/m/vfp-instructions, [Online; accessed 24-May-2023].

[62] ARM, ARM Compiler armasm User Guide Version 5.06, VFP Hardware, 2023, https://developer.arm.com/documentation/dui0473/m/overview-of-the-arm-architecture/vfp-hardware, [Online; accessed 10-March-2023].