Assessment of Edge Machine-Learning Systems under Radiation-Induced Effects

Matheus Garay Trindade

Université Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Nano Electronique et Nano Technologies (NENT)**

Arrêtée ministériel : 25 mai 2016

Présentée par

Matheus GARAY TRINDADE

Thèse dirigée par **Rodrigo POSSAMAI BASTOS**

préparée au sein du **Laboratoire Techniques de l'Informatique et de la Microélectronique pour l'Architecture des systèmes intégrés (TIMA)**
dans **l'École Doctorale Electronique, Electrotecnique, Automatique & Traitement du Signal (EEATS)**

Assessement of Edge Machine-Learning Systems under Radiation-Induced Effect

Thèse soutenue publiquement le **30 Septembre 2021**,
devant le jury composé de :

Dr. Rodrigo POSSAMAI BASTOS
Maître de Conférences, Université Grenoble Alpes, Directeur de thèse

Dr. Laurent ARTOLA
Ingénieur de recherche, ONERA, Examinateur

Pr Sergio CUENCA ASENSI
Professeur d'université, Universidad d'Alicante, Rapporteur

Pr. Alberto Bosio
Professeur d'université, École Centrale Lyon, Rapporteur

Dr. Patrick Girard
Directeur de Recherche, Université Grenoble Alpes, Président

"Quem tem um rumo de rancho pras quatro patas

Bota seu mundo na estrada batendo água."

Luiz Marenco

Acknowledgements

First, I would like to thank Dr. Laurent Artola, Dr. Alberto Bosio, Dr. Sergio Cuenca Asensi and Dr. Patrick Girard for agreeing to be part of my thesis committee. I would also like to thank you for all the remarks and questions that have been posed during the review and defense of the manuscript.

I would also like to thank my advisor Dr. Rodrigo Possamai Bastos, who has accepted me as his Ph.D. student and who throughout the development of the thesis has spared no efforts in guiding me and providing me with access to state-of-the-art radiation test equipment. I can say that I was a very privileged student when it came to radiation campaigns during my thesis.

Moreover, I would like show gratitude to the advisors that I had during my undergraduate studies at the Universidade Federal de Santa Maria. Dr. Osmar Marchi dos Santos, who accepted a very inexperienced 17 years old me into his research group and advised me throughout my five years of undergraduate studies. You have helped me even when I was in England, during my exchange year, helping me getting into a research group. When I came back, in no time I was placed by you in a project that has been one of the most exciting projects that I have participated: the Astros project. It was, as far as I know, the best project in the University to be at the time. I am really grateful for everything. Dr. Andrei Legg, you are also someone who was really important for me during my undergraduate studies. You were always there for everything that I needed and who helped guiding me on my final year graduation project (and there were some good laughs that came out of it!). Also, I remember when you came to me and said "I have heard of an opportunity in France, and while I do not want to lose you as a student, I believe it is a great opportunity for you and if you want to go, I will help with that the best I can". Thanks. It really meant a lot. I would also like to thank Dr. Simone Ceolin and Dr. Renato Machado, who have been of immense help during my time on the Astros project and who have trusted in my work. I really enjoyed working with you.

Ph.D. is hard, no questions about it, but it is easier when you have colleagues that make your day better, and I sure had that. Ricardo, you were much more than a colleague, you are real friend that I made during the thesis. And when I say real friend, I really mean it. We have published papers together, learned to ski together, went drinking beer together

and built a Viking city together (yeah reader, you heard it right). Whenever I needed, I felt that I could count on you, and I have no words for that. I hope that you know that you can always count on me; Thiago, you are another great friend that I have made on my thesis. You are probably one of the funniest people I have ever met, and it is amazing the amount of stuff that you have helped me with and I am really grateful for everything. The greatest thing is that I can continue working with you after my thesis!; Leonel, my first lab roommate! I can say that we had some good laughs and celebrations, especially when our team won the Libertadores; Rodrigo Iga, I really miss our discussions at coffee time and the great ski weekends that we have done together. Furthermore, I would like to thank Alexandre, Yoan, Mohammed, Gregoire, Jérémy, Nils, Assia, Liège and Tarso. You were great colleagues to have. I miss the environment at TIMA!

During my thesis, I have also made some really good friends outside the lab too. I would like to thank Luiza, you were the first person that I met (in a quite funny way) when I got in Grenoble and you have definitely made adapting to such a huge amount of change easier. You are a great friend. I would like to thank Katyanne. Is there a better guest than one that stays in your house and randomly cooks you a shrimp risotto? I also want to acknowledge Paola, Rafael, Raup, Natalia, Renato and Cocotas.

Good friendships can withstand distance and time. I want to acknowledge my friends (ODS!!!) from Santa Maria. I have met these guys when I was in school and while I now live an ocean away from them, whenever I step in Brazilian soil, they are eager to see me. Guto, I can honestly say that you are one the best friends that I have and I miss our long drives discussing politics, Grêmio, parties and plenty of legendary stories that we have been through, all of that while listening to some good ol' Pitbull. Rafael, man, we have done some stuff, right? Looking forward to a good barbecue that only the guys from the "border" know how to do. Bruno, curiously enough, you were one of the first people I met when I moved to Santa Maria, and we did not seem to have much in common, but we definitely become great friends. Beto, I really enjoy our discussions regarding a huge variety of games and I surely miss going out with you or even just chilling and playing some Magicka.

Finally, I would like to address the most important acknowledgements of this thesis. I would like to thank my parents, my dad Alcino and my mom Rosangela. I believe that to make justice of how thankful I am, I would need to write a whole other manuscript

explaining the multitude of ways that I am grateful. You have always encouraged me to do anything that I wanted. And really, ANYTHING. I believe that it must have been hard enduring me learning how to play the guitar, for instance! You pushed me to study foreign languages, and guess what, I am now using English and French every day, and both have been primordial for my studies. You have made sure that I would have access to the best education possible, and this has definitely played a major role in everything that I have achieved. Also, even though I was far away, I always felt safe and I never felt alone. I was always sure that you were on my side. I know that it is cliche, but I do have the best parents in the world and I love you. I would also like to thank my grandmother Maria, my cousins Rafaela, Maria Eduarda, Mariana and Fernanda, my uncle Pires and my Aunts Mariangela and Denise. I am grateful for all the good energy that you have sent towards me, for believing in me and for always receiving me so well when I visit you. An honorable mention as well is my dog Prince. He was a real companion that I had, who would curl up on my lap while I would study and try to cheer me up when I was feeling down.

It has been quite a ride. Now, let's see what is waiting for me in the future.

Contents

Introduction

Mankind marvels at *Artificial Intelligence* (AI) date back to as early as the antiquity. For instance, in the greek mythology, Talos is described as an automaton made of bronze that is tasked to protect Europa, the mother of King Minos, in the island of Crete. In the ancient Chinese *Liezei* Taoist book, an "artificer" builds a human-shaped automaton that could be mistaken by a live human. Mary Shelly's *Frankeintein* is an example of AI depicted in more recent literature, but that still pre-dates one of the biggest steps in mankind history that has since shaped our understanding (and expectations) of AI: the computer.

With the computational power yielded by the development of computers during the Second World War, computer scientists started to theorize the feasibility of true AI and what it should be like. The prestigious computer scientist Alan Turing even published a paper describing a method to identify if a machine is thinking: the famous Turing Test. Later advances in symbolic processing allowed by computers in the early 50s culminated in the recognition of AI as field with the Darthmouth Workshop in 1956. The years that followed were marked by great enthusiasm by the community, with a multitude of AI applications being developed, such as natural language processing and problem solving algorithms. However, due to hardware limitations, the development hit a barrier and, in 1974, started what is now know was "AI winter", a period where interest was decreased due to disbelief of community in the capabilities of such systems. While some spikes of interest have happened in the 80s, AI would only be regarded with enthusiasm again decades later.

Contemporary to the first developments in AI and later AI winter was the Cold War. This period was marked by constant dispute between the Western Bloc, i.e., Capitalist bloc, and the Eastern bloc, i.e., Communist bloc. These two blocs were in a constant arms race in a variety of domains in order to assert their political ideology and influence. Results of this dispute were, for example, proxy wars in what were known as Third World

countries, nuclear weapon development, and, notably for this thesis, the Space Race. Following the ballistic developments motivated by the second world war, one of the ways that both blocs started to display power was through space exploration. Notable achievements were the launch of the first artificial satellite (Sputinik 1) and the first human in space (Yuri Gagarin) by the *Union of Soviet Socialist Republics* (USSR) and the manned expeditions to the Moon (Apollo program) by the *United States of America* (USA). With the interest in spaceflight came many scientific and engineering questions. In this thesis, the focus is on the problem that was first described in [1]. The development of *Integrated Circuit* (IC)s started in the late 50s and were of great importance for space exploration. In fact, the ICs were used in the *Apollo Guidance Computer* (AGC), responsible for guidance and control of the Apollo spacecrafts. However, as stated [1], the space environment was not only hazardous to humans, but to ICs as well. Communication satellites were showing unexpected behaviors as their circuitry was triggering in an anomalous fashion. The authors, through an experiment using an electron microscope, showed that ionizing particles could indeed provoke unintended triggering of the IC, effectively causing soft-errors, i.e. non-permanent errors in which bits of the system have their value inverted (bitflips). They also showed that the expected error rate extracted from their experiments agrees with the error rate observed in a satellite in orbit, corroborating the claim that the faults were caused by cosmic rays.

From that point on, it was evident that space posed an additional challenge due to the way it affects electronics. This raised the question of whether such effects could happen inside the Earth atmosphere. Later, in 1996 [2], E. Normand showed that, in fact, the same effects observed in space could happen at ground level. The difference resides in the type of particles that cause these effects. Earth's atmosphere and magnetic field work together to shield the planet from cosmic radiation. However, when cosmic rays collide with particles in the Earth's atmosphere, neutrons are released at different levels of energy. These neutrons are capable of producing bitflips, just as cosmic rays.

While research on the effects of radiation developed through the late 70s and 80s, AI was stagnated and still facing some disbelief. It was not until the 90s that the enthusiasm would came back. Developments in semiconductor technology made computers reach a new height in terms of processing power. This allowed for some major breakthroughs, such as the development of an AI that beat the chess grandmaster Garry Kasparov [3]. In

the early 2000s, also pushed by the developments in processing power, ML has started to gain more notoriety. ML is a branch of AI in which the focus is to develop algorithms that learn from data. As more and more people started to gain access to the internet, the amount of usable data skyrocketed opening many opportunities for ML applications. For instance, it was at this time that web companies started to analyze their user's data to provide personalized experiences, i.e., the first development of recommender systems. Furthermore, various entities promoted competitions to instigate researchers and encourage the development of ML applications. For instance, the company Netflix, at the time an online DVD-rental company - held an open competition, namely The Netflix Prize, in which the goal was to predict the rating a user would give for a movie based on the previous ratings he/she has assigned to other movies. Also at this time, The American *Defense Advanced Research Projects Agency* (DARPA) held multiple self-driven cars competitions [4].

During the 2000s and the 2010s, as technology was getting more and more ubiquitous, so was machine learning. During this time, making calls became just minor functionalities of smartphones as they evolved into very capable devices. Home automation devices, such as Amazon Alexa and the Google Home, reached the selves. Even websites evolved into personalized experiences tailored towards each of its user. ML was behind all these advances. However, these solutions often require hefty processing power, which these devices cannot provide. The solution, in these cases, is to forward the data to be analyzed to the cloud. While this solution is acceptable in these cases, not all applications can rely on a connection to a server. For instance, self-driven cars relying on internet connection would be disastrous. Thus, there has been movements towards edge-AI. The idea is to embedded the ML computation to the devices closer to the sensors that capture data, i.e. the ML is placed on the edge. Works have been conducted to port the models over to platforms that are candidates to for edge processing, such as *Graphic Processor Unit* (GPU)s, *Field-Programmable Gate Array* (FPGA)s and microcontrollers.

Edge-AI has also started to find its way in space applications. The Perseverance Mars rover, launched in July 2020 and which landed on Mars in February 2021, implements Edge-AI. Embedded in the rover is the *Planetary Instrument for X-ray Lithochemistry* (PIXL), an equipment designed to look for traces of ancient microscopic life. AI models are used to better point the instrument to the intended location [5]. The rover also used

AI to help on its landing. This is especially important as the distance between Mars and Earth makes that a signal takes from 5 to 20 minutes to travel between the planets, making it impossible to pilot the craft remotely from Earth. Apart from the rover, companies and universities have been investigating the usage of AI in nanosatellites. An example is the Lockheed Martin joint project with the University of Southern California for the development of the La Jument nanosatellites, specifically for testing ML implementations in space.

Critical systems are systems in which a failure may lead to catastrophic or costly results. For instance, a failure in a self-driven vehicle may cause the loss of a human life. A failure on a rover on Mars may turn billions of dollars in investment into a very expensive robotic decoration to the landscape of Mars. As ML is being embedded in such systems, and they are subject to radiation induced faults, be it at ground level, avionic altitudes and on space, it is important to understand how these faults affect these systems. In this sense, the objective of this thesis is to evaluate the reliability of machine learning models implemented in different hardware platforms.

In this work, the reliability of three popular ML models against radiation induced faults was studied. These models are: *Artificial Neural Network* (ANN), SVM and *Random Forest* (RF). To estimate their reliability, two approaches were used. The first was fault emulation. As mentioned, radiation induced faults may result in bitflips. To evaluate how these bitflips affect the algorithms, they are implemented on a device and the memory of such device is manipulated to artificially invert the values of bits. In doing so, it is possible to evaluate the impact of such faults on the system while also identifying which parts of the system are the most sensitive. The second method used was accelerated testing. As the time expected for a fault to happen is long, the target devices containing the ML algorithms are placed in front of artificial radiation sources that surpasses natural fluxes by various orders of magnitudes. This way, months or even years of real-life system exposure is simulated in a fraction of the time, allowing for an estimation of fault rate and behavior of the system in the real environment. In terms of the devices used, two SVM models were implemented using FPGA. Furthermore, implementations of the three ML models in microcontrollers were also studied. All these solutions were also tested using fault injection as a complimentary study.

This thesis is organized as follows: Chapter 1 describes the terminology related to the

radiation testing of components and systems. Chapter 2 contains a review on ML and the algorithms used in this work. Chapter 3 presents the first evaluation of the reliability of a SVM under the effect of fast neutrons. The algorithm was implemented in FPGA and tested using a fast neutron source and fault emulation techniques. Chapter 4 complements this work by testing two SVM models implemented in FPGA under the effect of thermal neutrons by also performing an accelerated test using a radiation source and a fault emulation campaign. Chapter 5 analyses the effect of fast neutrons in microcontroller implementations of ANN, SVM and RF using a custom fault emulator. In Chapter , the same ML algorithms are tested using a neutron source. Chapter 7 introduces a custom fault emulator implemented to serve as tool in this thesis. Chapter 8 presents the final remarks.

1

Radiation testing of Components and Systems

Contents

In this Chapter, a review on radiation testing of components and systems is presented. The terminology and definitions used in this thesis are discussed.

1.1 Soft Error Taxonomy

As defined in the JESD89A standard [6], soft errors are any type of non-destructive functional disruption of a system normal execution induced by any type of radiation. A taxonomy of soft errors is present in Figure 1.1. Each type of Soft Error is described in the sequence.

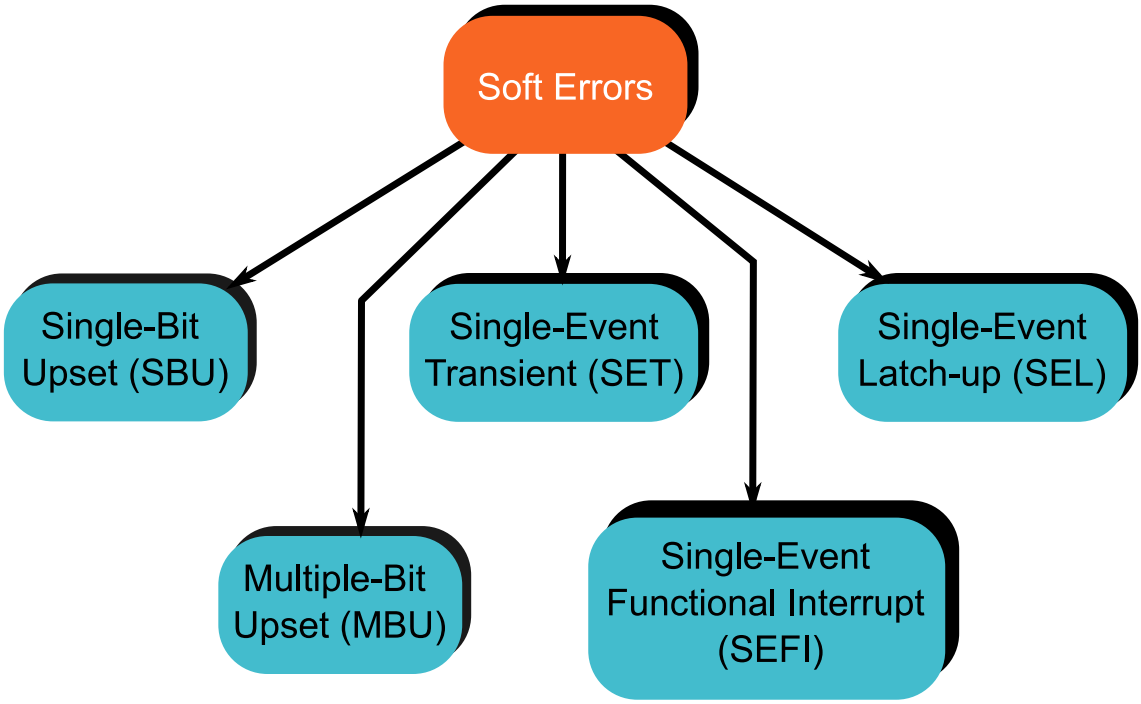


Figure 1.1: Taxonomy of Soft Errors, according to [6]

1.1.1 Single-Bit Upset (SBU)

A Single-Bit Upset (SBU) is when a particle causes a bit of a memory element to change from its expected. Examples of memory elements are Random Access Memories (RAMs), latches and registers. It is worth nothing that no permanent damage is done to the components, i.e., the memory still operates correctly but it holds an erroneous value. If the erroneous value is overwritten before being used, the system will not present an unexpected behavior.

1.1.2 Multiple-Bit Upset (MBU)

With the ever-increasing miniaturization of devices, memory cells are getting physically close to each other with every new technological step. This makes so that a single particle may be capable to cause bit upsets in multiple memory cells, causing a Multiple-Bit Upset (MBU). It is important to notice that MBU refers to a single event causing multiple bits to flip, and not multiple events each causing a bit to change.

1.1.3 Single-Event Transient (SET)

When a particle hits a semiconductor device, it may transfer energy to it, causing transient pulse in the circuit. In analog systems, this can be very problematic, as disturbances in a signal may cause the system to fail, e. g. the pulse may be amplified by an operational amplifier. In digital system, SET may interfere with the combinational logic of the circuit. This disturbance may be capture by a memory element, possibly leading to unwanted behavior [7].

1.1.4 Single-Event Functional Interrupt (SEFI)

A *Single Event Functional Interrupt* (SEFI) is a soft error which causes the system to lose its functionality. For example, a soft error may attack a control bit, making the system reset, hang or malfunction. The system must be recoverable without power cycling for the event to be considered a SEFI. Also, no permanent damage incurs in the circuit.

1.1.5 Single-Event Latch-up (SEL)

Similar to a SEFI, a *Single Event Latch-Up* (SEL) is a soft error that causes the system to lose its functionality, but different than a SEFI, it is only recoverable through power cycling, i.e. turning the system off and back on. It happens when a particle passes through a sensitive part of the circuit, creating an abnormal high-current state. As noted in [6], a common example is when a particle induces the creation of parasitic bipolar (p-n-p-n), effectively shorting power to ground.

1.2 Metrics

In this Section, the metrics related to radiation testing are presented and discussed.

1.2.1 Particle Flux

The particle flux (ϕ), commonly referred to simply as flux in the context of radiation testing, is the rate in which particles reach a component per unit of area. As so, it is expressed in $particle/s/cm^2$. In whichever physical environment a component is executing, be it ground level, avionic altitudes or in space, it is constantly being bombarded with a spectrum of particles at multiple energies, each environment having it is own characteristics. For instance, in space, cosmic rays contribute the most to particle spectrum, which in turn makes the flux of protons, alpha particle and heavy ions the highest. At ground and avionics level, due to Earth's natural protection, the flux of cosmic ray-related particles is lower. However, secondary particles are generated when cosmic rays collide with oxygen and nitrogen atoms in the atmosphere releasing neutrons, which are in turn the most common particles at ground and avionics altitude [8].

Knowing the particle flux of an environment allows for the design of experiments to simulate it. Particle accelerators are great tools to not only mimic a environment (even though partially) but to perform accelerated testing. They can provide fluxes that are orders of magnitude higher than natural fluxes, allowing for the emulation of months (or even years) of the real environment. For example, for some of the experiments described in this work, the GENEPi2 [9] 14 MeV accelerator from the Laboratoire de physique subatomique et de cosmologie de Grenoble (LPSC) has been used, in which we have used fluxes of roughly 10^6 $neutrons/s/cm^2$. The natural flux of 14 MeV neutrons at ground level is roughly $5 * 10^{-2} neutrons/s/cm^2$ [10], with the GENEPi2 flux being $2 * 10^7$ times larger than it. Thus, 1 hour of testing in the GENEPi2 equipment is equivalent to 2.281 years in the real environment.

1.2.2 Fluence

Fluence Φ represents the total number of particles per cm^2 that have traversed an environment in a time interval Δt. It is represented as the integral of the flux over time, described in Equation 1.1:

$$\Phi = \int_{t_0}^{t_f} \phi(t)dt \tag{1.1}$$

where $\Delta t = t_0 - t_1$ are the initial and final time of the interval being measurement, respectively, and $\phi(t)$ represents the flux as a function of time. As it is a product between flux ($particle/s/cm^2$) and time, the result measuring unit is $particle/cm^2$. Fluence is particularly useful to define further metrics, such as cross-section *Failure in Time* (FIT), to be described in the sequence.

1.2.3 Cross-section

Cross-section is a useful tool to quantify the radiation effects on a hardware platform or system. It describes the probability of a single particle causing a failure in the system. Thus, it is expressed as shown in Equation 1.2:

$$\sigma = \frac{N_f}{\Phi} \tag{1.2}$$

where N_f is the number of failures identified during the irradiation with the fluence Φ. Assuming that a fault is a result of a single particle interacting with the component, the measurement unit for the cross-section is cm^2.

It is common to distinguish between static and dynamic cross-sections. Static cross-section measures the sensitivity of a hardware platform agnostic of an application. For example, static cross-section of a *Dynamic Random Access Memory* (DRAM) memory is the probability of a particle to cause a bit flip in a cell. In this case, a way to measure the static cross-section is to load a pattern on the memory, irradiate it, count the number of failures and use Equation 1.2. On the other hand, dynamic cross-section represents the probability of a particle to cause a fault when a application is being run on a platform, e. g. a machine learning algorithm running on an FPGA. It takes into account that the application itself may mask some faults, e.g., faults outside the memory space being used may not need to a failure, providing a metric that represents the system as whole.

While the dynamic cross-section is the most useful when it comes to the final implementation of system. Measuring the dynamic cross-section is expensive due to the cost of radiation campaign. To overcome this, it is possible to estimate the it from the static cross-section, which may be provided by the platform vendor. They can be related by

Equation 1.3.

$$\sigma_{dynamic} \approx \sigma_{static} * \theta_{system} \tag{1.3}$$

with θ_{system} being the probability of a fault becoming a failure, which can be estimated through fault injection campaigns.

1.2.4 Soft Error Rate (SER)

Soft Error Rate (SER) is the rate of soft error observed in a system in a given environment [7]. A popular way to express it is through FIT. 1 FIT is equivalent to 1 failure per 1 billion hours. It is calculated using Equation 1.4:

$$FIT = \sigma * \phi_{env} * 10^9 \tag{1.4}$$

where σ is the system cross-section and ϕ_{evn} is the environment flux expressed in $particle/cm^2/h$. 10^9 represents 1 billion hours, as per the definition of FIT.

2

Revision on Machine Learning

Contents

In this Chapter, the ML terminology and techniques are described. Furthermore, a review of the algorithms studied is presented.

2.1 Definitions

2.1.1 Dataset

A dataset is a collection of data of a particular problem. An example of a famous dataset is the Fisher's Iris flower dataset [11]. The dataset is a collection of measurements, sepal and petal length and width, of multiple specimen of Iris flowers, a flower genus, of three different species: *Setosa*, *Virginica* and *Versicolor*. The data was collected by E. Anderson and became famous when R. Fisher [11] used statistical learning techniques to design a classifier capable to predict the species of an Iris flower from its measurements. Other examples of datasets are shown in [12].

A dataset may be labelled or unlabeled. A labeled dataset contains the expected output of its samples. For instance, the Fisher Iris dataset, the class of each specimen is known. This allows for the training of machine learning models to be used to infer an output over a sample whose class is unknown. On the other hand, an unlabeled dataset is not annotated. This type of dataset is often used for clustering problems. Figure 2.1 shows an example of labeled and unlabeled datasets.

2.1.2 Input Sample

An input sample is a single instance that follows the structure of the dataset of which some information is unknown and a machine learning technique is going to be used to predict this information. For example, for the Fisher Iris dataset, an input sample is the measurements of a flower whose class is unknown. The goal of the machine learning algorithm is to predict the class of this input sample.

2.1.3 Feature

A feature is one of the dimensions of a sample. For example, as mentioned, the Iris flower dataset is a set of measurements of multiple specimens of Iris flowers. These

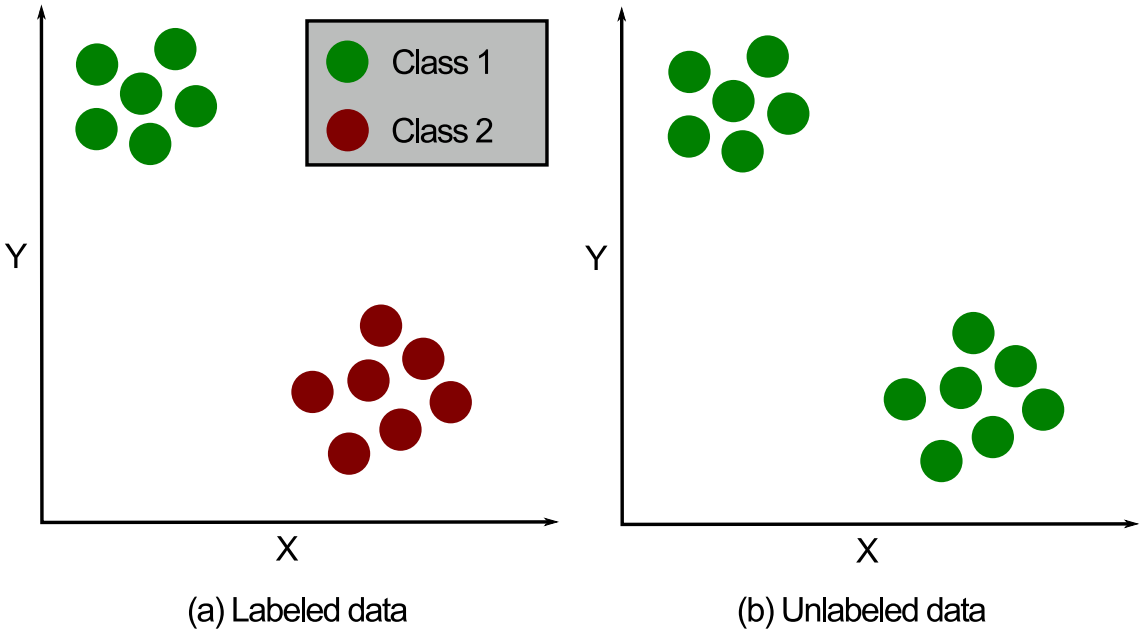


Figure 2.1: (a) Illustrative example of a labeled dataset. (b)Illustrative example of an unlabeled dataset.

measurements are sepal length, sepal width, petal length and petal width. Each of these measurements is a feature, i.e., each sample is composed of 4 features/dimensions.

2.1.4 Supervised Learning

Supervised Learning is a subset of the machine learning algorithm that performs the learning over a labeled dataset. The workflow of supervised learning algorithms is often divided in two distinct phases: learning and classification. During the learning phase, a training algorithm to generate a mathematical model that maps samples of a dataset to their expected labels. The dataset used for this learning phase is referred to as training dataset. In the classification phase, the generated model is used to classify samples whose labels are unknown. For instance, for self-driving cars, a dataset of road pictures (samples) associated with the action that should be taken, such as breaking or turning (label) could be used to train a model. On a real-life environment, a camera could be mounted on a car, input pictures to the model (input samples), which would output the action that should be taken. Please note that this is an oversimplification of the self-driving car application and it is used only with illustrative purpose. An illustration of a supervised learning is shown inf Figure 2.2-(a).

2.1.5 Unsupervised Learning

Different than supervised learning, unsupervised learning algorithm performs learning on unlabeled datasets. For examples, a machine learning design may have access to a dataset of emails and want to develop a model that is capable of distinguishing between spam and regular mail. However, the dataset is unlabeled, meaning that the mails are not identified as spam or not. To goal of an unsupervised learning algorithm is to identify the underlying structure of data and group them accordingly, i.e., find clusters of data. Unsupervised learning is illustrated in Figure 2.2-(b).

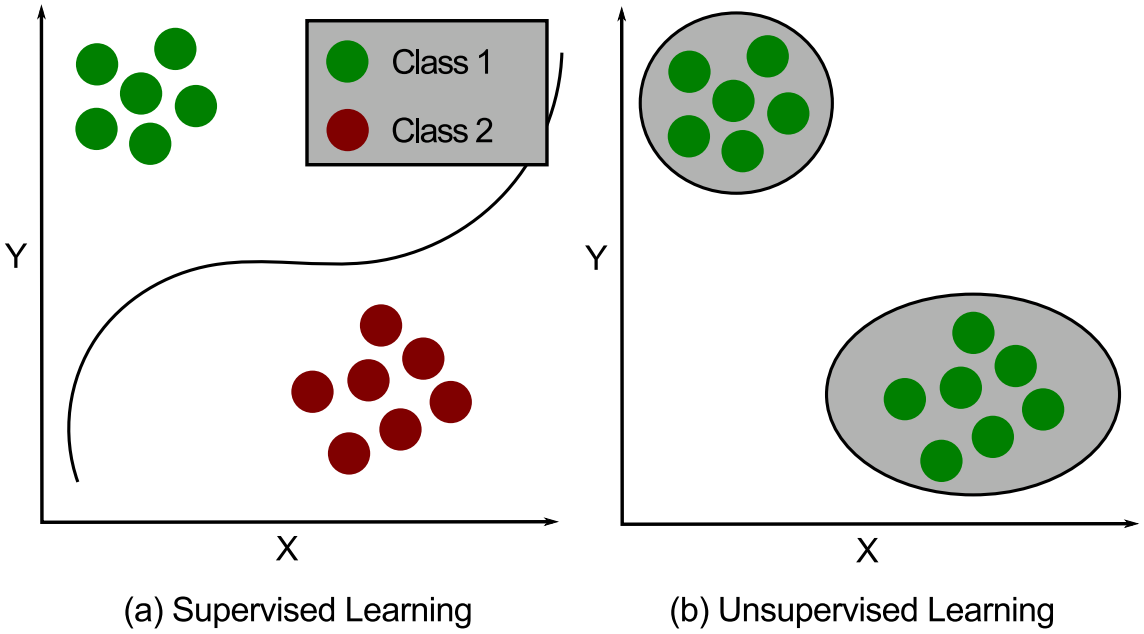


Figure 2.2: (a) Illustrative example of a labeled dataset. (b)Illustrative example of a unlabeled dataset.

2.1.6 Classification

The goal in a classification problem is to derive a function $X \rightarrow Y$, s.t. $X \in \mathbb{R}^n$, n being the number of features in the input dataset, Y being a discrete set. For instance, the Iris Flower problem [11] is an example of classification problem. The output set Y is defined as $\{setosa, virginica, versicolor\}$. In a more semantic definition, a classification consists predicting the class of an unknown sample based on its characteristics.

2.1.7 Regression

A regression problem, similar to a classification problem, also consists in finding a $X \rightarrow Y$ function, s.t. $X \in \mathbb{R}^n$, n being the number of features in the input dataset, but differently than classification, Y is a continuous value. An example is the Boston housing problem [13]. The dataset contains information concerning housing in the Boston Mass, U.S.A. The goal is to estimate the value of real-state based on this information. As "value" is a continuous variable, this is a regression problem.

2.2 Artificial Neural Networks (ANN)

ANN is a classical machine learning technique, with its first dating to the 40s, with the work of McCulloch and Pitts [14]. In their paper, the authors mention that due to the characteristics of biological neurons, their behavior could be treated as propositional logic. At that moment, the idea was embraced by the community and other related works exploring the technique were developed. However, the research on the topic reached a plateau at the end of the 60s due to limited processing power at the time. The interest resurged in the mid-80s [15] with the rise of parallel processing and, in recent years, their popularity only grew, being implemented in platforms such as GPUs [16, 17] and FPGA [18].

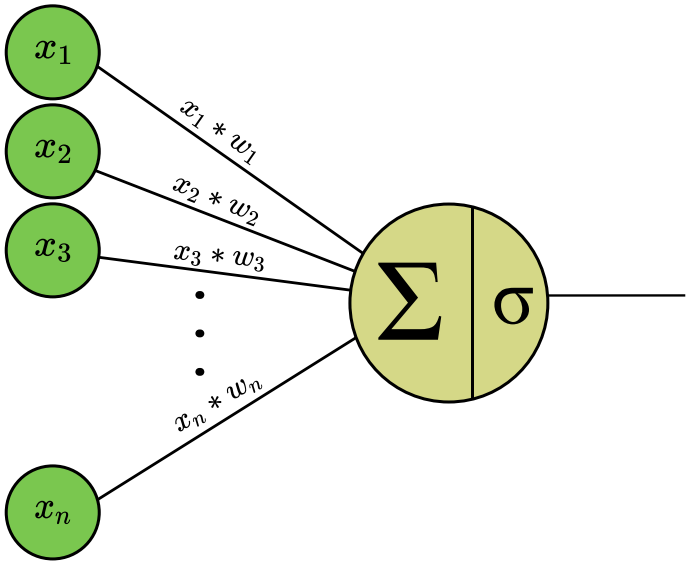


Figure 2.3: Structure of a Perceptron/Neuron

As mentioned, the idea of the ANN is inspired on the way biological neurons be-

have. In biology, neurons receive a series of inputs (neurotransmitters) on their dendrite, and, depending on what was received, forwards neurotransmitters over its axon. The analogous "artifical" neuron is shown in Figure 2.3. The neuron receives a set of inputs $\{x_1, x_2, ..., x_n\}$. Each input is multiplied with a weight $\{w_1, w_2, ..., w_n\}$ and the results are summed. This process is also known as *Multiply-ACcumulate* (MAC) operation. The result is then evaluated on an activation function σ. A structure composed of a single neuron as portrayed in Figure 2.3 is called a Perceptron. While limited, a Perceptron works a binary classifier, i.e., a classifier capable of distinguishing between two classes. For example, let's take a dataset with only two classes $\{A, B\}$. A perceptron can be trained to output positive values for samples of the class A, while outputting negative values for samples belonging to the class B by carefully selecting a proper activation function. A possible candidate to achieve this behavior is a using the $\sigma(x) = tanh(x)$.

As discussed, the perceptron is somewhat limited, being restricted to binary problems. Furthermore, it was shown in [19] that a perceptron is incapable of learning the XOR function. To overcome this, the idea of multi-layer networks was introduced. The neurons are organized in a layered manner, as shown in Figure 2.4. The general idea is that a neuron is going to receive the outputs of the neurons in the preceding layer, perform the MAC operations and forward to the next layer, hence why this topology is also known as feed-forward neural network. The structure is divided in three segments: the input layer, the hidden layers and the output layer. The neurons in the input layer are solely responsible to inject the inputs to the neurons of the next layer. The number of neurons in this layer is exactly the number of dimensions, i.e., features, of the dataset. The majority of the learning often takes place in the hidden layers. The number of layers as well as the number of neurons per layers are variable, to be defined the neural network designer. The output layer, similar to the input layer, has its size tied to the dataset being used. The number of neurons is exactly the number of classes of the problem at hand, each neuron specifically representing one of the classes, which makes a neural network intrinsically capable of performing multiclass classification.

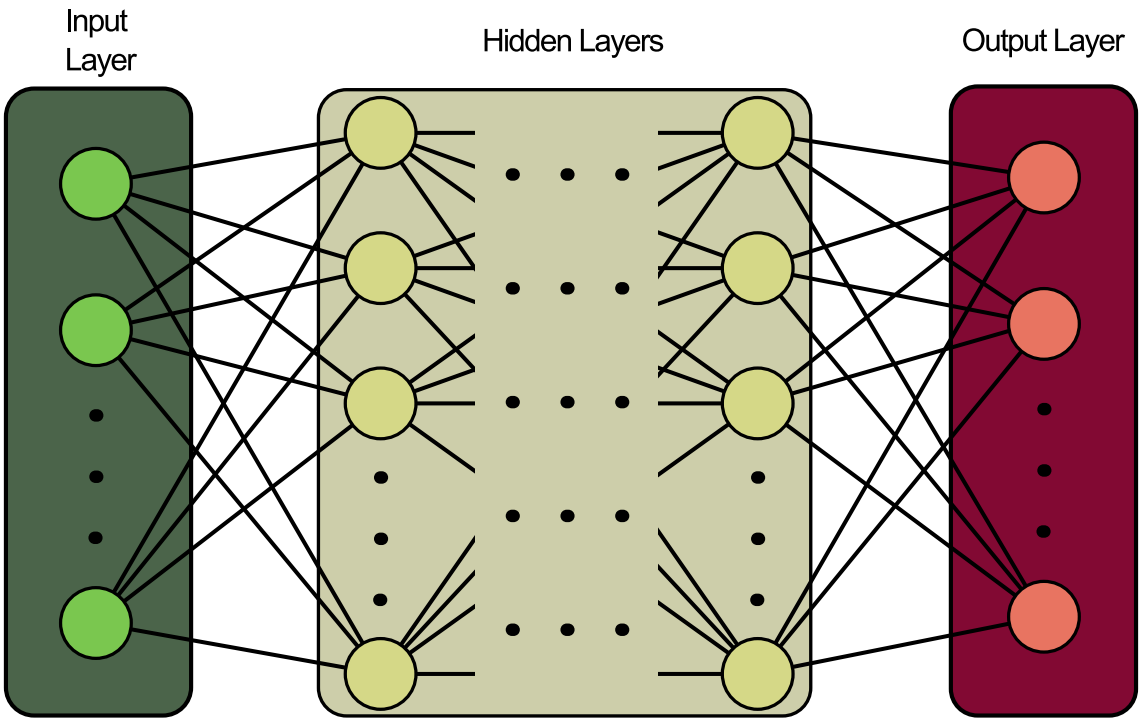


Figure 2.4: Structure of a Artificial Neural Network

2.3 Support Vector Machine (SVM)

SVM is a classification algorithm belonging to the group of supervised machine learning techniques [20]. The algorithm addresses the problem of binary classification, i.e., a problem in which an observation (herein an input vector) has to be classified in one of two possible classes. Being a supervised machine learning technique, its workflow requires two phases, each one performed with a different algorithm: one for training *Sequential Minimal Optimization* (SMO) algorithm) and another for classification (SVM algorithm).

Figure 2.5 presents an application of a SVM-based classification able to indicate whether an astronaut reaches a risky condition to have a cardiac problem – for instance, according to her/his heartbeat rate while moving or speeding on a treadmill in a space station.

An input vector for the SVM algorithm is a pair $(heartbeatrate, speed)$, which are referred to as features or dimensions. The set of all measured pairs $(heartbeatrate, speed)$ is defined as a feature space. In order to train the SVM algorithm, i.e. to model the SVM algorithm equation, a set of input vectors – herein called training vectors – is required. The training vectors are indeed a subset of the feature space, however, the class of them are known beforehand, e.g., input vectors from people whose cardiac conditions are known.

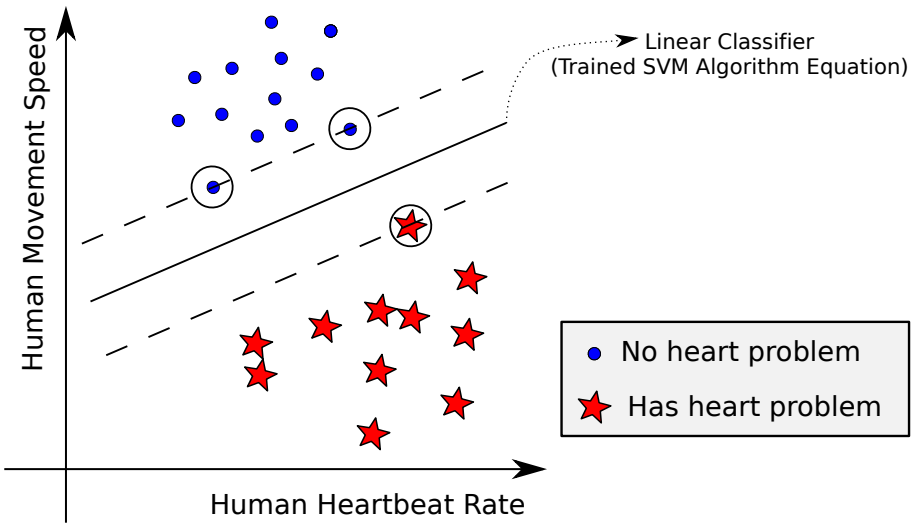


Figure 2.5: An SVM algorithm equation (linear classifier) trained to classify the heartbeat condition. The horizontal axis represents the human heartbeat rate, while the vertical axis represents the human movement speed.

At the training phase, the SMO algorithm uses the training vectors to calculate the weights of the linear function that better separates input vectors from the two classes, for example in Fig. 2.5: class “No heart problem" and class “Has heart problem", and the training vectors respectively represented by blue dots and red stars. The calculated weights, therefore, model a linear classifier dividing the elements of the two classes, i.e. they model the SVM algorithm equation.

Formally, a tuple $(\vec{x}_i, y_i)$ defines a training vector, in which the support vector $\vec{x}_i \in \mathbb{R}^2$ in the example: heartbeat rate and speed features – both belonging to the set of real numbers – that are measurements performed on a person i. Moreover, the class $y_i \in \{-1, 1\}$, and in the example: -1 and 1 represent respectively the class “No heart problem" and the class “Has heart problem".

At the classification phase, the trained SVM algorithm is able to classify an input vector whose class is unknown, e.g., if the heartbeat rate and speed of an astronaut running on a treadmill indicate either “No heart problem" or “Has heart problem". The SVM algorithm equation to classify an input vector $\vec{x}$ with an unknown class is thus defined as:

$$score(\vec{x}) = \sum_{i=1}^{n} \alpha_i y_i (\vec{x}_i \cdot \vec{x}) + b \tag{2.1}$$

Each α_i is a weight, found at the training phase and associated with its respective training vector $(\vec{x}_i, y_i)$. The weights α_i shape the linear classifier (i.e. the SVM algorithm

equation). At the training phase, the SMO algorithm also calculates the bias factor b. The sign of the $score(\vec{x})$ determines the class of the input vector $\vec{x}$, in the example: positive for “Has heart problem" and negative for “No heart problem".

2.3.1 Multiclass SVM

Even though, originally, SVM was created to perform binary classification. However, it may be extended to fit multiclass problems. This is achieved by dividing the dataset into subsets so it is possible to use binary classification. The two most popular techniques are One-vs-One and One-vs-All. In this work, we made use of One-vs-One, which is explained in the sequence. One-vs-All was left out for the sake of brevity.

In the One-vs-One approach, a binary classifier is trained for each pair of classes. For instance, if there are three possible classes, e.g. $\{A,B,C\}$, a classifier is trained only with samples from classes $\{A,B\}$ in order to classify unknown samples between classes A and B. Another one is trained with samples from classes $\{A,C\}$ to classify an unknown simple into either A or C. A classifier for $\{B,C\}$ is also trained. To infer a class for an unknown sample, it is evaluated on the three trained SVMs, each one inferring one class. The class that is the most inferred is chosen as the final class. For example, if SVM $\{A,B\}$ outputs A, SVM $\{A,C\}$ outputs A and SVM $\{B,C\}$ outputs B, as A has been inferred twice whereas B was only inferred once and C has not been inferred, the final class is A

2.4 Random Forest

RF is a supervised learning algorithm that performs the learning by building a collection *Binary Decision Trees* (BDT). To understand its model, it necessary to discuss the workflow of BDTs.

The idea of a BDTs is to partition the training dataset in two, trying to group elements of the same class together. The same process is then applied to each of these partitions recursively, up until the partitions have only examples of a single class. This process can be represented using binary trees, as illustrated in Figure 2.6. Each non-leaf node contains the partitioning function, while the leafs contain only an identifier representing the class of the elements it contains. To classify an input sample using a BDT, the input sample

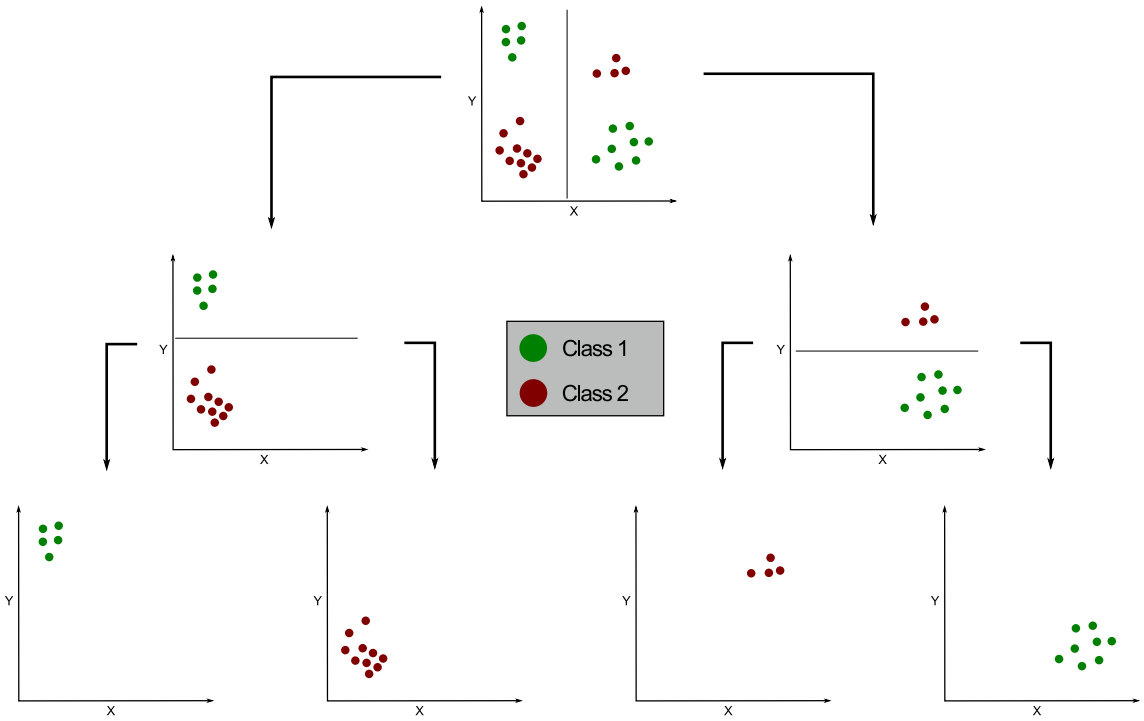


Figure 2.6: Example of partitioning performed by BDT

is evaluated in the partitioning function of the root node, which will direct it to either the left or right child node. This process is repeated recursively until a leaf is reached. The class assigned to the input sample is the class represented by the leaf reached.

While BDTs work by themselves as supervised learning algorithms, they are prone to overfitting the data. As mentioned, the partitioning is done until they contain members of a single class. This is particularly problematic for datasets that contain outliers, i. e. samples of class that differ from other elements of the same class, or noisy data. The algorithm will try to accommodate these examples in the partitioning process, which may create a BDT that does not generalise well over the training dataset. A way to try to overcome this is building a "forest", i.e. a collection of BDTs. Each tree is trained over slightly modified versions of the dataset generated using bagging, also known as bootstrap sampling. Each new dataset is generated by performing uniform sampling with replacement on the original dataset. The number of datasets and, consequentially the number of BDTs trained is defined by the machine learning designer and varies from case to case. To classify an input sample using the forest generated, it is first classified all the trees generated then, the most common class among the results is the final classification of the RF.

3

Support Vector Machine under Radiation Effects

Contents

3.1 Introduction

Machine learning algorithms have been increasing in popularity in both academic and industry because of their capacity to learn from existing data, and to predict future outcomes. Those two features, learning and predicting, have motivated the use of this type of algorithm in many applications such as medical diagnostics [21], robot intelligence [22], and geoscience/aerospace domain [23]. What all these application have in common is the need to identify a pattern – in a raw amount of data – and to decide which action has to be taken based on the pattern classification.

Among machine learning algorithms, the SVM, described in Section 2.3, has been extensively used over the past years for pattern recognition and data mining. The SVM has high generalization capacity and robustness in both data classification and regression. SVMs are more commonly used in classification problems because they are based on the idea of finding a hyperplane that best divides a dataset into classes. In other words, SVM separates the elements of a dataset in classes that have similar properties. However, since the amount of data are increasing in size and complexity, the necessity of providing some alternatives to accelerate the performance of the SVM classification is still a current issue.

In the sense of accelerating the SVM classification, this algorithm has been implemented in FPGAs [24]. Nevertheless, FPGAs are considered sensitive to radiation effects [25], especially *Static Random Access Memory* (SRAM)-based FPGAs on which particle-induced transient faults may corrupt their configuration memory in a effect also known as *Single Event Upset* (SEU)s. Transient faults are indeed able to change three elements in an FPGA: the configuration of a routing connection; the configuration of a *Look-Up Table* (LUT); and the data inside of an embedded *Block RAM memory* (BRAM). As the effects might be persistent, a new bitstream must be loaded to the FPGA to correct it. Furthermore, a transient fault is able to invert a *Flip-Flop* (FF) of a *Configuration Logic Block* (CLB) of the user's sequential logic. In this case, the next load of the FF may correct this fault.

Recent works have studied the effects of radiation-induced transients faults on hardware implementations of machine learning algorithms, such as ANNs and Convolutional Neural Networks [26–28] and Bayesian machines [29]. To the best of our knowledge, however, no related work has been studied the radiation effects on SVM. In this Chapter,

we present the first assessment of an FPGA-implemented SVM architecture under radiation effects by looking at how induced transient faults affect its behavior. An extensive fault emulation campaign and experimental radiation tests with a 14-MeV neutron generator beam have been carried out to analyze SVM responses. The radiation test campaign has been performed in GENEPi2 neutron accelerator [9] using an Artix-7 FPGA.

3.2 Case-Study SVM Architecture

Depending on the application, a software-implemented SVM algorithm is continuously demanded to classify input vectors $\vec{x}$, consuming energy and occupying computational resources that could be reallocated to other tasks. A hardware-implemented SVM algorithm is an effective alternative solution to accelerate the computation and to save the energy of systems.

Training and SVM-based classification phases have been implemented in hardware as an *Application-Specific Integrated Circuit* (ASIC) in [30, 31] and by using an FPGA in [24, 32] to speed up the SVM algorithm execution. Furthermore, related works have also implemented in hardware only the SVM-based classification phase [31] [24], conducting the training phase in software platforms such as MATLAB or LibSVM [33], and enabling further energy savings in ASICs used to make preliminary offline inference.

3.2.1 State-of-the-Art SVM in Hardware

Several solutions have been proposed for implementing SVM classification in hardware [24,30,31]. The implementation in [31] multiplies the *Support Vector* (SV)s with an input vector all in parallel. This has been done to achieve maximum performance to suit their application (voltage-droop prediction), which demands a very fast classification time. The limitation of this approach is being able to generate only linear and second-order polynomial classifiers. To overcome this limitation, works [24] and [30] have implemented non-linear classification by using a *COordinate Rotation DIgital Computer* (CORDIC) algorithm, which requires solely hardware-friendly functions (e.g., shift and sum operations) to approximate the calculations over some clock cycles. Likewise in [31], the operations with the SVs and an input vector are also performed in parallel in [24] and [30]. The

CORDIC algorithm-based implementations have higher memory requirements in comparison with [31] because look-up tables are used to approximate exponential functions and more clock cycles are needed to output an answer. However, they generate more complex classifiers.

3.2.2 SVM Architecture Design

As our dataset is linearly separable, further described in Subsection 3.2.3, we opted for the approach in [31] with a first order, i.e. linear, classifier to explore its benefits in terms of memory requirements and performance. Figure 3.1 shows the case-study SVM architecture. In addition, we have calculated the $\alpha_i y_i$ products beforehand, reducing one multiplication for each SV $\vec{x}_i$, and further optimizing the SVM architecture. The final implementation is completely combinatorial. The circuit is composed by three main parts: the Multipliers, the Adders and the Output, as illustrated in Figure 3.1.

The SVM architecture adopted a 16-bit fixed-point representation, 8 bits being dedicated to representing the real part, following the ideas in [24,30,31]. Through simulations, we have confirmed that the representation is enough to avoid possible overflows and also to maintain sufficient precision. The primary inputs are composed of 32 bits (16 for each dimension of the input vector). The primary outputs are composed of 49 bits to maintain the calculation precision.

3.2.3 Set of Input Vectors

The target set of input vectors (dataset) has been obtained from Monte-Carlo simulations representing current peaks and global delays obtained from golden ICs and faulty ICs [34]. The input vector is 2-dimensional, and 150 input vectors have been obtained from golden IC samples and 150 input vectors from faulty IC samples. The dimensions are thus:

- **Dimension 1**: global delay
- **Dimension 2**: current peak

This set of input vectors is sufficient to distinguish faulty asynchronous IC samples from fault-free asynchronous IC samples [34]. The set of input vectors has been parti-

tioned into 2 subsets of the same size, one for training and another for classification, each one with 75 golden IC samples and 75 faulty IC samples. A SVM model has been generated by using MATLAB. From this model, we have obtained the α's and their respective support vectors $\vec{x_i}$. In total, 50 support vectors $\vec{x_i}$ have been generated at the training phase.

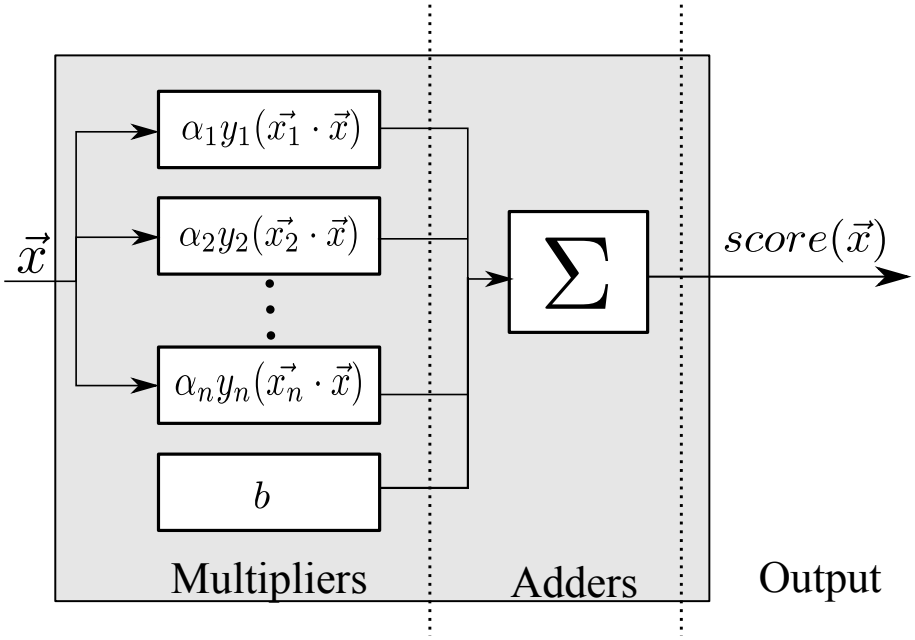


Figure 3.1: Overview of the hardware-implemented SVM architecture design.

3.3 SVM Architecture Assessment Through Fault Emulation Campaign

This section assesses the ability of the case-study SVM architecture to tolerating transient faults. We have used a fault-injection method able to emulate transient faults in an FPGA-implemented SVM architecture.

3.3.1 Device Under Test (DUT)

The DUT is a fully combinational SVM architecture design using fixed-point representation for its support vectors $\vec{x_i}$ and wights α_i (cf. section 3.2). The target platform is a Zynq-7000 [35], which is composed of two main parts: the Processing System (PS), consisting of an ARM Cortex-A9, and the Programmable Logic (PL), a Xilinx Artix-7 FPGA. The Zynq-7000 has been used on the ZedBoard development board The SVM

architecture was implemented in PL part by using VHDL. The resource utilization of the PL is shown in Table 3.1.

The SVM algorithm depends fundamentally on multiplications, cf. section 2.3. Most modern FPGAs, including the Artix-7, have Digital Signal Processing units (DSPs), which implement multiplications in dedicated hardware. All the multiplications of the DUT have been mapped in the DSPs, which is the reason why DSP resources are prominent in Table 3.1.

Table 3.1: Resource utilization of the PL (Artix-7)

Resource	Utilization	Elements
Flip-Flops (FFs)	1.65 %	1751
Digital Signal Processing units (DSPs)	40%	88
Look-Up Tables (LUTs)	7.24%	3854

3.3.2 Assessment metrics

We have assessed the case-study SVM architecture under transient faults by stimulating different input vectors $\vec{x}$ at its primary inputs, observing the related results $score(\vec{x})$ at its primary outputs, and classifying them into three types:

- **No Failure:** the result at the primary outputs of the SVM architecture does not differ from the golden reference (i.e. fault-free output vector);
- **Tolerable Failure:** there is a mismatch between the result at the SVM primary outputs and the golden reference, however, the resulting class is correct, i.e. the sign of the result at the primary outputs is equal to the sign of the golden reference;
- **Critical Failure:** there is a mismatch between the result at the SVM primary outputs and the golden reference, and the resulting class is not correct.

In order to assess the rate of critical failures provoked by the emulation of a single transient fault on a node n of the SVM architecture design, we define the following metric:

$$CriticalFailureRate(n) = \frac{\#\,CriticalFailures(n)}{\#\,InputVectors} \tag{3.1}$$

in which # $CriticalFailures(n)$ is the total number of critical failures provoked by a single transient fault injected at node n of the SVM architecture design, and # $InputVectors$ is the total number of vectors tested at the primary inputs of the SVM architecture with the node n under the same fault emulation.

3.3.3 Fault Emulation Method

Figure 3.2 illustrates the workflow proposed to apply the fault emulation method on SVM architecture. The method emulates transient faults based on an in-house tool proposed in [36–38]. Initially, the HDL description of the SVM is used to obtain the first synthesis in Step ①. In this step, the synthesis of the SVM is targeted for a specific FPGA (an Artix-7 in our proposed architecture). We consider Synplify FPGA synthesis software as it allows exporting a netlist, which is necessary in further steps.

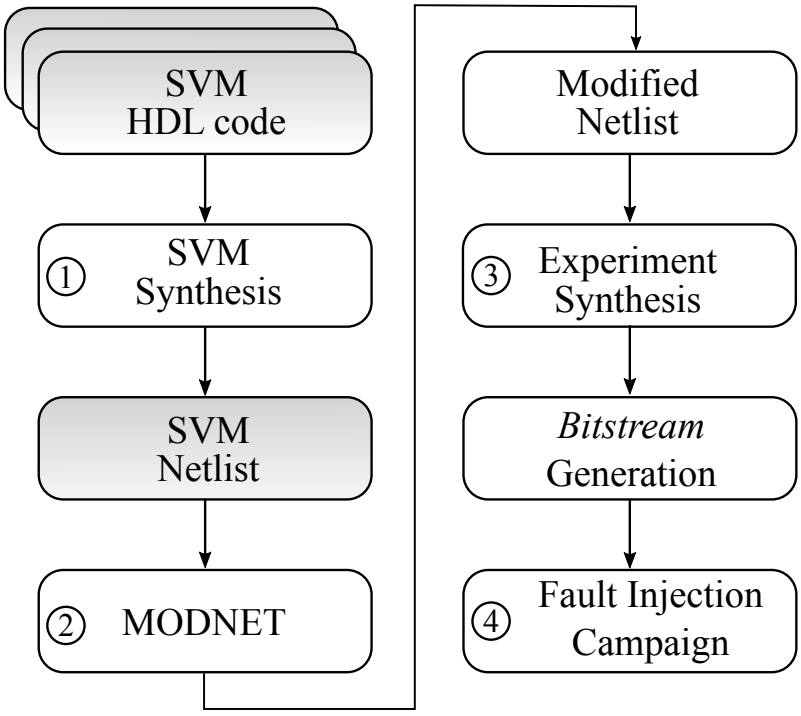


Figure 3.2: Workflow of the FPGA-based fault emulation method

In Step ②, the netlist is used as input for the *MODify NETlist* (MODNET) tool [38]. The output of MODNET is a modified (but functionally equivalent) Netlist with a large number of extra input signals used to access all memory cells and logic blocks of the design to inject faults in the SVM. The resulting synthesis of the modified netlist includes some additional combinational circuitry to the design.

MODNET tool adds a multiplexer at the outputs of the LUTs and logic gates of the netlist to either enable faults at these nodes or disable them by using the signal *INJ*, cf. Figure 3.3.

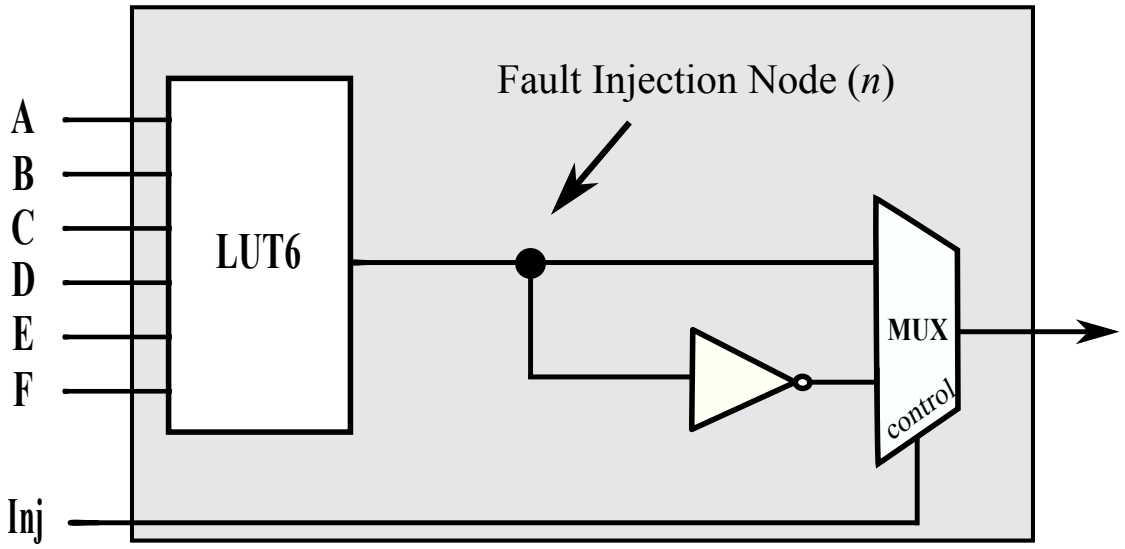


Figure 3.3: Structure of the fault injection node

In Step ③, a campaign controller is integrated within the modified Netlist for the target FPGA. The campaign controller is implemented in a soft-core processor that is in charge of managing the transient fault emulation campaign by being directly wired to the SVM architecture modified by MODNET. To this end, the netlist obtained in Step ② is synthesized in the *Electronic Design Interchange Format* (EDIF) and then attached to the processor via a direct interface. The resulting *bitstream* implementing the complete circuit (controller and SVM) is thus generated and implemented in the target FPGA.

Finally, the FPGA-based experiment in Step ④ can be directly executed from the soft-core processor without requiring additional or external hardware support. Indeed, the whole fault emulation campaign (including the post-processing of the results), are conveniently encoded in the processor software. By accessing high-speed interfaces connecting the SVM, the software executes several iterations of fault-injection experiments with different data inputs and fault nodes. The MicroBlaze processor uses an UART to communicate with the outside to report the status of the experiment and its final results.

In summary, Figure 3.4 shows the framework of the method that we have applied to perform the fault emulation campaign, which is integrated into a single FPGA where the SVM and the experiment controller are instantiated and connected by an *Advanced eX-*

tensible Interface (AXI). The MicroBlaze uses this communication channel to configure the transient fault emulations in the components already intervened by MODNET (*INJ* signals), as well as to configure the input and read the output values of the SVM. In this particular case, the MicroBlaze processor (campaign controller) is a master, whereas the SVM is a slave. The results of the fault emulation campaign are presented in subsection 3.3.4.

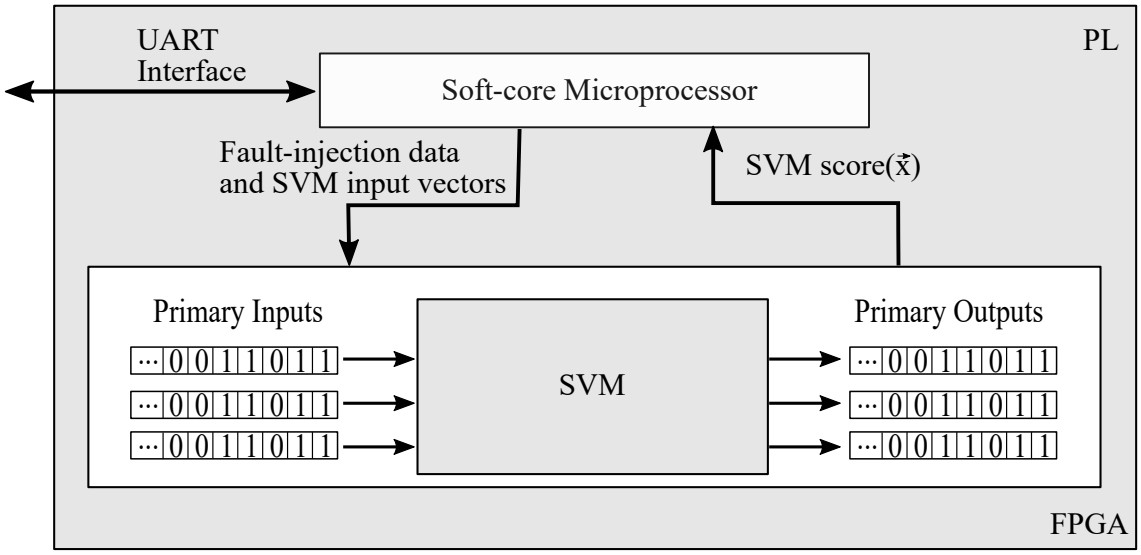


Figure 3.4: Framework of the method used to perform the fault emulation campaign.

3.3.4 Results of the Fault Emulation Campaign

The fault emulation campaign has been configured to extensively analyze the behavior of the SVM architecture in the presence of faults. The MODNET tool (step ② in Figure 3.2) has identified in the SVM architecture 1350 nodes to be assessed through a transient fault emulation campaign. For each node, a fault has been injected successively for the set of 150 input vectors. It is important to note that the node holds the fault over the entire clock cycle. For each fault emulation, the primary outputs of the SVM architecture have been compared with the correct result (golden), and each observed failure has been logged in the campaign controller.

It is worth mentioning that 58.8% of the faults have been injected on the outputs of the LUTs connected at the DSP outputs (the Multipliers), while 29.3% of the faults have been injected on the outputs of the LUTs that are in charge of computing the Adders part. The rest of the faults (approximately 11.9%) have been injected on the outputs of the

LUTs used by the configuration signals of the SVM architecture. After an extensive fault emulation campaign covering several nodes of the SVM architecture, encompassing a total of 202,500 faults injected in the SVM architecture, results show that more than 29% of emulated single transient faults provoked a critical failure in the SVM architecture. It means that 71% of the fault emulations led to either tolerable failure or no failure. Figure 3.5 presents the histogram of the rates of critical failures (Equation 3.1) provoked by the emulation of single transient faults on the nodes of the SVM architecture design. The results indicate that most nodes are quite sensitive to transient faults (critical failure rates between 20% and 60%), and only a small number of them has a critical failure rate higher than 60%. Such considerable critical failure rates on several nodes suggests that the fault masking effects are low due to the parallel configuration of the operators and the short path between the primary inputs and outputs of the SVM architecture. Selective mitigation techniques could be applied to make more robust such nodes with high critical failure rate.

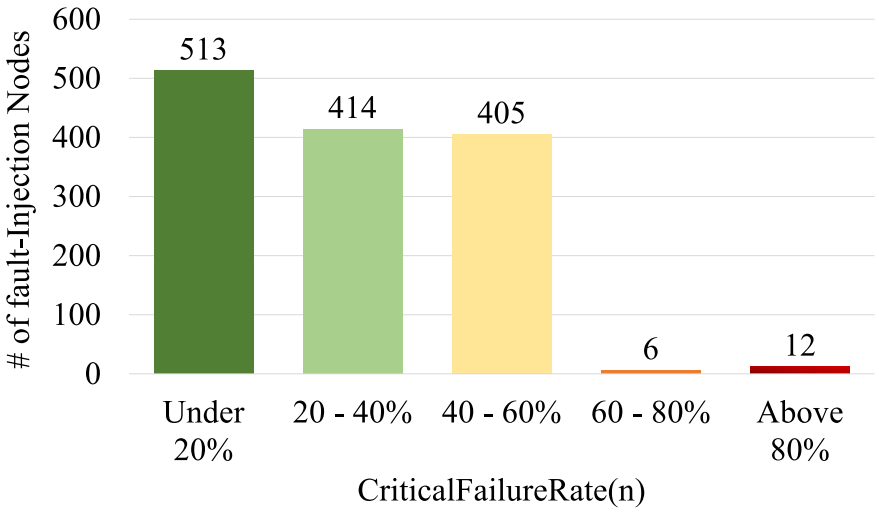


Figure 3.5: Histogram of the critical failure rate of the injection nodes on the SVM architecture as given by Equation 3.1

Figure 3.6 shows the correlation among the most critical failure rate nodes and their relative position in the SVM's circuitry. It is worth mentioning that those nodes were extracted from the FPGA LUTs, and their relative positions are logically represented into the Figure 3.1. The results plotted in the Figure 3.6 correspond to the 18 nodes with a $CritiaclFailureRate > 60\%$. As indicate in the figure, the most critical faults are distributed in the SVM architecture. In regard to the multipliers, all the eight critical

components are positioned in the first logical level of the multiplier, i.e., they are LUTs that receive the first computation of the DSPs with most significant bits. Analyzing the region of the Adders, we can see that eight other faults/nodes can be classified as critical. Unlike the multiplier results, where critical faults are significantly positioned in the first logical level of the SVM architecture, failures in the Adders are diffused in the circuit. It means that the critical failures can occur in different stages of the combinational Adders' circuitry. When the fault injections are performed in Output, only faults happening close or on the actual sign bit cause critical failures. Even though the most susceptible LUTs are spread out in the architecture, they can be mapped. The very small number of LUTs with a high *CriticalFailureRate*, as shown by Figure 3.6, suggests that hardening could be done on these LUTs to improve the SVM fault tolerance with low area overhead.

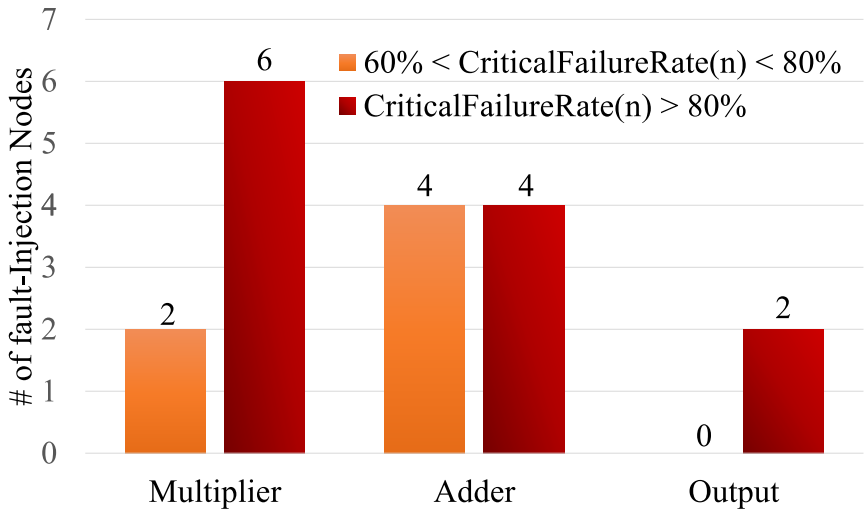


Figure 3.6: Histogram representing the correlation among the most critical failure rate (Equation 3.1) nodes and their position relative to the SVM's circuitry implemented in a FPGA.

3.4 Radiation Test Experiment and Results

This section describes the radiation experiments conducted with a neutron source and an analysis of the obtained and assessed results.

3.4.1 Radiation Test Set-Up

A radiation campaign has been performed at the GENEPi2 neutron source, at the *Laboratory of Subatomic Physics & Cosmology* (LPSC), in Grenoble (France) [9]. Unlike the LANSCE neutron beam, which ranges from 10-MeV to 750-MeV having a flux that is 10^6 higher than the natural neutron flux of this spectrum at 40,000 ft. [10], the GENEPi2 is a 14-MeV neutron generator with a maximum flux that exceeds the natural flux of 14-MeV neutrons at 40,000 ft. by a factor of 10^{10}. The board was irradiated for 6 hours and 45 minutes, yielding a total neutron fluence of $1.944 * 10^{10}\ n * cm^{-2}$ and an average flux of $8 * 10^5\ n * cm^{-2}/s$. Note that we did not use the maximum flux of the accelerator. with an environment temperature of 18°C. A polyethylene shielding with a hole was placed in front of the ZedBoard to protect parts other than the Zynq-7000.

The architecture used for the radiation test has a minor difference from the one used in Section 3.3 and is shown in Figure 3.7. While in the fault emulation campaign the FPGA has been driven by MicroBlaze, in this radiation test experiment we adopted the ARM as controller for the SVM. However, the SVM architecture design on the PL remained unchanged. In order to reduce the number of input bits sent from the ARM to the SVM architecture, we instantiated a small controller that had 32-bit wide to store the set of input vectors tested on the FPGA.The controller fetches the data from the memory and forwards it to the SVM. It then retrieves the output and exposes it through an AXI interconnect. Figure 3.8 shows a picture of the test set-up.

The ARM processor provides the controller on the FPGA with the index of the input vector to be tested, as the input vectors are stored in a ROM in the PL, and reads the result. It then forwards the result through the serial port. The L2 cache of the ARM processor has been disabled to reduce the probability of faults affecting the PS [39].

3.4.2 Radiation Test Method

Figure 3.9 presents the method we have used during the radiation test experiment. The set of input vectors is continuously running in the FPGA. The radiation is able to alter the configuration SRAM, which contains the FPGA bitstream, i.e. it is able to create an error. This error may then lead to an alteration of the SVM calculation structure, mathematically changing the classification function. The score of an input vector evaluated in this

faulty SVM may deviate from its expected result, i.e., we analyze the primary outputs of the SVM architecture, and then we compare them with a golden reference. We use this property to identify when an error has happened. Whenever an error has been identified, we rerun the entire set of input vectors on the faulty SVM to classify each input vector according to the metrics in Subsection 3.3.2, and finally, we log the results. When it is done rerunning the input vectors, the FPGA is reset to erase the error, and the process is restarted. It is worth noting that the errors detected during the radiation test could be either from SEUs or Single Event Multiple Upsets (SEMU), the latter becoming more important in recent technology nodes. A SEMU happens when a single particle affects more than one adjacent memory elements, which is becoming more likely with the shrinking of transistor size [40].

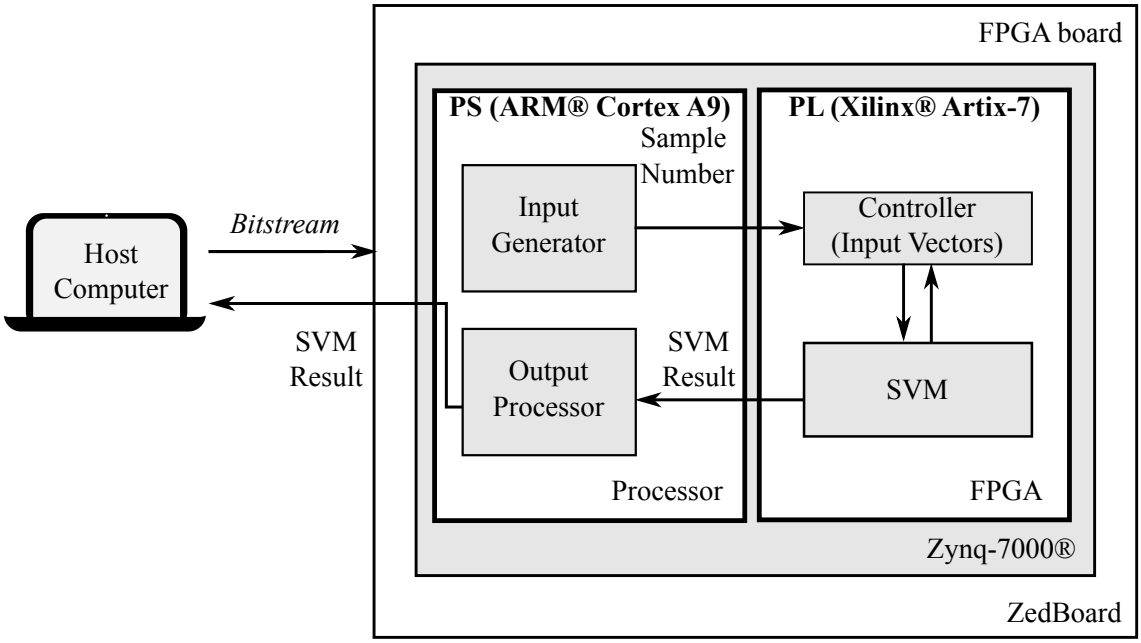


Figure 3.7: Zynq-7000 set-up under radiation test

3.4.3 Assessment of Radiation Test Results

During the neutron radiation test campaign, we have identified 13 errors, of which 2 crashed the FPGA and 11 errors that allowed it to continue to produce results. Even though crashes have been responsible for 15% of the total number of errors, they are not related to the case-study SVM architecture design but due to a fault in the device performing the serial communication with the control computer. The obtained static cross-

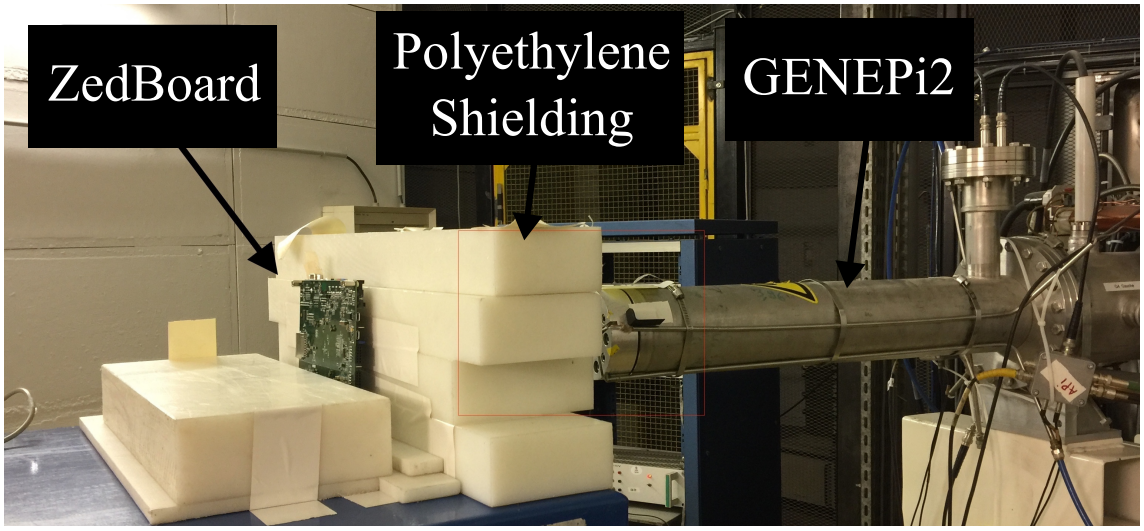


Figure 3.8: FPGA board installed at the GENEPi2 accelerator neutron facility

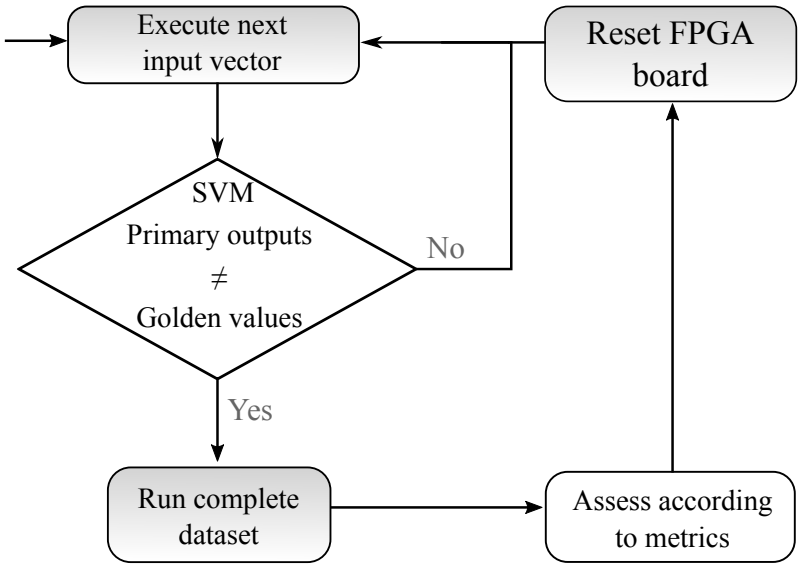


Figure 3.9: Method used on the radiation test

section and the FIT are respectively $5.65 * 10^{-10}$ cm^2 and 7.91, considering New York's 14 $n/(cm^2 \cdot h)$ neutron flux at sea-level [2].

A total of 1650 input vectors have been evaluated on SVMs with an altered behavior due to errors, as the 150 input vectors have been input to the SVM when an error has been identified. Of those, 1168 continued to output the expected result (*No Failure*), 92 have been considered as Critical Failure and the remaining 390 as Tolerable Failures. As shown in Figure 3.10, only 5% of the evaluated input vectors have resulted in Critical Failure. When there is an error, there is a 95% chance that the error will not lead to a misclassification. The full report, showing the classification of each input vector on each error is shown in Figure 3.11. Each column in this figure represents an input vector evalu-

Table 3.2: State-of-the-art of Machine Learning test under radiation effects

Work	ML Algorithm	Dataset	Errors Provoking Tolerable Failure	Integrated System	Techn. Node	Event
[27]	ANN	Iris Flowers	64%	FPGA (Artix-7)	28nm	Neutrons
[26]	ANN	Iris Flowers	65%	FPGA (Artix-7)	28nm	Heavy Ions
[27]	CNN	MNIST	~74%	FPGA (UltraScale+)	16nm	Neutrons
[28]	CNN (YOLO)	MNIST	92%	GPU (Tesla K40)	28nm	Neutrons
[28]	CNN (YOLO)	MNIST	82%	GPU (Titan X)	16nm	Neutrons
This	SVM	Fault Detection	73%	FPGA (Artix-7)	28nm	Neutrons

ated on one of the errors identified during the test. The colors represent the classification of the error according to the metrics in Subsection 3.3.2. Only one of the errors caused the majority of the input vectors to be misclassified. This has been possibly an error very close to the primary output. This error has forced every primary output to a negative score, effectively classifying every input vector to the same class. However, the majority of the errors did not seem to deviate the classification function by much, as the vast of the input vectors remained classified correctly, which suggest a good level of tolerance to errors built-in in the algorithm.

In terms of errors, Figure 3.12 presents a report of the radiation-induced errors that provoked tolerable and critical failures in the SVM architecture during the radiation campaign. As shown in Figure 3.11, 3 of 11 errors caused at least one of the input vectors to be misclassified. This indicates that 27% of the errors identified during the radiation test caused the SVM architecture to produce a critical failure. This result suggests that the case-study SVM architecture indeed has a level of intrinsic fault tolerance, however, more information is needed to better identify the nodes that cause critical failures.

On an FPGA-designed SVM architecture, errors may change a x_i, a α or the calculation logic shown in Equation 2.1. These reshape and/or dislocate the linear classifier. If

there is an error on one of the less significant bits of a x_i or α, the separator displacement may be so small that most of the input vectors are still classified correctly, even though their score, i.e. the output from the classification function, changes. Mathematically, the algorithm is an accumulation (the Adders in Figure 3.1) of the dot product of the support vectors and the input vectors each weighted by an α (these operations performed by the Multipliers illustrated in Figure 3.1). If a fault happens on the Multipliers, depending on the bit, the result would not change much. As the result is an accumulation of the complete set of support vectors, a small change in one does not interfere much, e.g., an input vector with a golden score of 1.8 is classified with a score of 1.5, still being classified correctly. The same is true for the Adders. Changes in the least significant bits are less likely to compromise the algorithm as they are not likely to accumulate to a point of disturbing the sign bit. It is worth pointing out that different input sample will suffer differently. An input sample that is close to the linear classifier, e.g., score of 0.2, will be more easily misclassified as a small perturbation can already make it negative. This indicates that the distribution of the input samples is important to the intrinsic fault tolerance of the algorithm. Our classes are well separated, which made our samples have high scores, corroborating with the fault tolerance. In addition, The output is composed of 49 bits representing the score. While the 48 least significant bits are part of the calculation, they are irrelevant to final result, as only the sign bit indicates the class. This indicates already a high level of fault tolerance. Thus, even faults that occur very close to the output of the system may be tolerated.

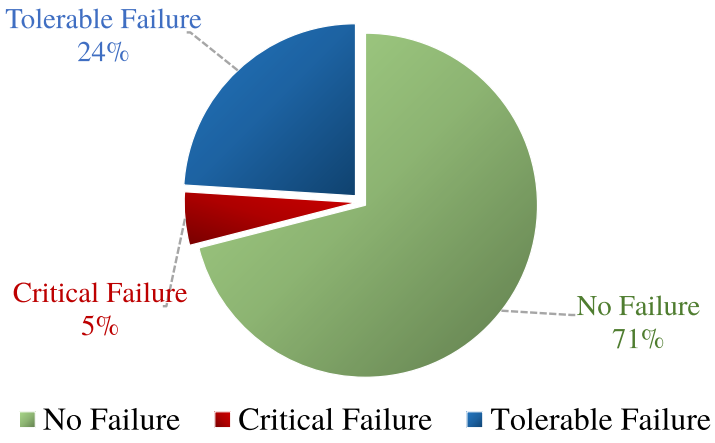


Figure 3.10: Percentages of the total number of situations originated by the 11 neutron radiation-induced errors that have been detected provoking either a failure (critical or tolerable) or no failure

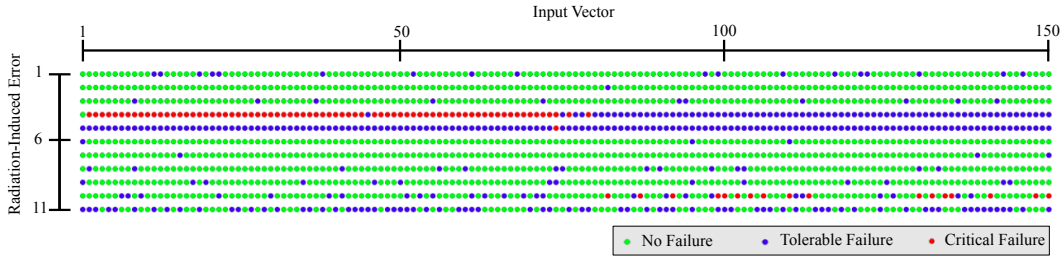


Figure 3.11: Map of the failures provoked by the neutron radiation-induced errors in function of the input vectors $\vec{x}$ that have been tested. The row numbers (1 to 11) represent the labels attributed to the radiation-induced errors and the column numbers (1 to 150) represent the labels attributed to the input vectors. Each color point means if the radiation-induced error provoked a critical failure (red point), a tolerable failure (blue point), or no failure (green).

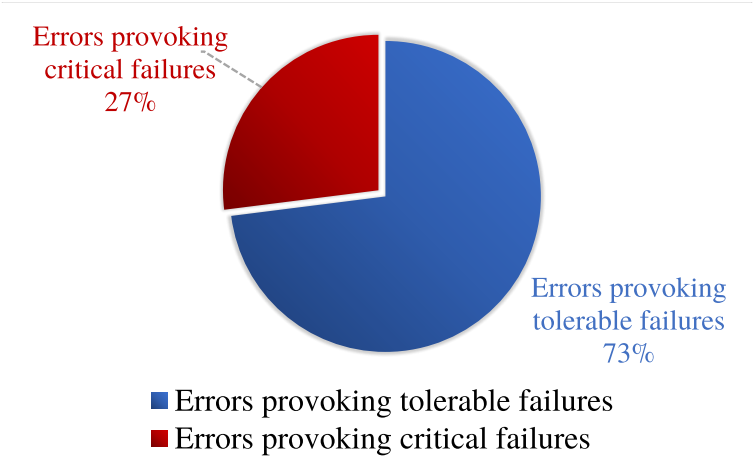


Figure 3.12: Percentages of the 11 neutron radiation-induced errors that have detected provoking either critical failures (at least one) or tolerable failures (at the worst case)

The most harmful effects are modifications on the most significant bits of the support vectors $\vec{x_i}$, weights α_i, or changes in the operators (as the technique is implemented in an FPGA). Only one of the 11 non-crashing radiation-induced errors led the majority of the input vectors to be misclassified (critical failure). This indicates that, for the case-study SVM architecture, this type of error is less likely to happen. This compliments the results found during the

Results from both radiation test and fault emulation campaigns suggest that a hardware-implemented SVM technique does have an intrinsic fault tolerance, as identified in other similar experiments with different Machine Learning algorithms [26, 29]. It is worth pointing out that no fault-tolerance mechanisms have been included in the case-study SVM architecture.

3.4.4 Comparison with State-of-the-Art Works

Evaluation of machine learning algorithms under radiation effects is still a new topic with few works published. In [26], the authors evaluated the intrinsic fault tolerance of Artificial Neural Networks (ANN) implemented in FPGA under Heavy Ions. In [26], the authors investigate the fault tolerance of both ANNs and *Convolutional Neural Network* (CNN), a variant of ANN which is very popular in image processing application, also on FPGA but evaluated under neutron effects. In [28], the authors evaluated the fault tolerance of CNNs in GPUs. These works are summarised in Table 3.2.

Our implementation achieved a higher percentage of errors provoking tolerable failure when compared to the ANNs implemented in the same FPGA [26, 27], the Xilinx®Artix-7, which suggests that the SVM may have an higher intrinsic fault tolerance when compared to ANNs. Furthermore, it also has a fault tolerance comparable to the CNN in [27] implemented in an FPGA in a 16-nm technology. As mentioned in Subsection 3.3.4, the applied dataset may have an impact in the fault-tolerance. A more complete investigation is to be further investigated. When comparing to GPU implementations of CNNs, while GPUs present better fault tolerance, FPGA implementations cannot be ruled out as they may be faster for some applications [41].

3.5 Conclusions

Radiation test results provided in this work suggest that the target FPGA-designed SVM architecture is robust to most single transient faults we have studied. However a considerable number of radiation-induced errors (more than 25 %) is able to provoke critical failures. Previous works on other machine learning techniques, such as the Bayesian Machines [29] (not evaluated under radiation) and ANNs/CNNs [26–28] have also shown that these techniques do have a built-in fault tolerance. The contribution of this Chapter is the assessment of the intrinsic fault tolerance of a SVM architecture. This is of great interest, especially when it comes to safety-critical applications.

4

Effects of Thermal Neutron Radiation on a Hardware-Implemented Machine Learning Algorithm

Contents

4.1 Introduction

The effects of radiation induced transients have been studied in multiple hardware implementations of machine learning algorithms. ANNs and Convolutional Neural Networks have been analyzed in [26–28], while [29] focused on Bayesian Machines and the work in Chapter 3, published in [42], on Binary SVM. In this Chapter, the literature (and the work in Chapter 3) is extended by studying the effects of thermal neutrons on hardware implemented SVMs by assessing the effects of radiation in Multiclass SVMs. In this work, we first-handedly investigate these effects by conducting a thermal neutron radiation campaign and an extensive fault injecting campaign on a Binary and a Multiclass SVM to better understand how the radiation affects them. Furthermore, we compare the results of a Multiclass SVM to the ones of a Binary SVM. The radiation test campaign has been performed using the D50 thermal neutron source at the Institut Laue-Languevin [43]. The fault injection campaign was based on partial configuration of the SVM bitstream. Both campaigns made use of a Zynq-7000 *System-on-Chip* (SoC) as test vehicle.

4.2 Case-Study SVM Architectures

In this Section, we present the SVM architectures used for this work. Two SVMs were implemented, a binary and a multiclass. The SVM algorithm is described in Section 2.3. The design choices follow those of the work presented in Chapter 3, discussed in Section 3.2. For both SVMs, only the classification step was implemented.

4.2.1 Binary SVM Architecture Design

The binary SVM design used in this work is the same used in Chapter 3. Its more in-depth description is presented Subsection 2.3. The dataset used was also the same, described in Subsection 3.2.3, but, in order to better exercise the circuit, 116 randomly generated samples were added to the dataset after training was done.

4.2.2 Multiclass SVM Architecture Design

As discussed in Subsection 2.3.1, a Multiclass SVM is composed by a collection of Binary SVMs aggregated by a voter. In this work, each Binary SVM was implemented following the description in Subsection 3.2.2, using the same value representation. The voter was not implemented in the FPGA, outputting the score of all the SVMs. It is up to the client application to evaluate the score and perform the voting. The designed circuit is also fully combinational.

4.2.3 Set of Input Vectors for the Multiclass SVM

For the Multiclass SVM, the dataset chosen was the Iris flowers [11]. It is originally a 4-dimensional dataset that contains 150 samples of three different species of Iris flowers (50 of each): setosa, virginica and versicolor. Only two dimensions have been kept for this experiment as they hold enough information for training a performant SVM classifier. The dimensions are:

- **Dimension 1**: Petal length
- **Dimension 2**: Petal width

This dataset has been partitioned in a training set and a classification set, with 75 samples (25 of each species) each. The training set has been used to train a One-vs-One Multiclass SVM, yielding three Binary SVMs, 2 SVMs with 2 Support Vectors $\vec{x_i}$ and one with 16 Support Vectors $\vec{x_i}$. Following the idea used in Subsection 4.2.1, 116 randomly generated samples were added to better cover the architecture designed.

4.3 SVM Reliability Assessment Through Emulated Fault Injection

Emulated fault injection was used in this work to cross-validate results from radiation experiments and further investigate areas of improvement in the *Design Under Test* (DUT).

Compared to emulated fault injection, accelerated irradiation experiments provide a better approximation to the use of the DUT in real environment and can provide a more comprehensive test coverage reaching all relevant structures of the integrated circuit. However, radiation experiments are less powerful in pinpointing the DUT submodules more susceptible to SEUs that could be candidate to mitigation strategies leveraging DUT reliability. This section describes the fault injection methodology and the results obtained.

4.3.1 Fault Injection Set-up

The test vehicle used in fault injection is a ZedBoard development board which is equipped with a Xilinx Zynq-7000 MPSoC hosting the DUT. The Zynq-7000 device is divided in two main parts that are a dual core *Processing System* (PS) based on Arm® Cortex®-A9 and a SRAM-based FPGA *Programmable Logic* (PL) that, for this device part number, uses technology equivalent to a Xilinx 7 Series Artix-7 FPGA.

The SVM computation core, implemented in VHDL, is wholly hosted in the FPGA (PL) side of the Zynq-7000. A small part of the application, implemented as software in C language, is hosted in the ARM (PS) side of the device and is responsible mostly by coordination and reporting tasks, not playing relevant role in the computation effort. However, between these two parts of software and SVM core, there is a communication infrastructure, based on AMBA AXI interface, with some modules also implemented in the FPGA side through the use of parameterizable *Intellectual Property* (IP) design blocks provided by the FPGA manufacturer. The resource utilization and frequency of operation for both the Binary and Multiclass SVMs are shown in Table 4.1.

	Binary	Multiclass
Frequency	133 MHz	133 MHz
LUT	7%	5%
LUTRAM	1%	1%
FF	2%	2%
BRAM	1%	1%
DSP	40%	28%
IO	1%	1%

Table 4.1: Resource utilization of Zynq-7000 for the Binary and Multiclass SVMs.

That accessory communication infrastructure using prebuilt IP blocks is relevant to fault injection because, while implemented in FPGA, it is also susceptible to SEUs, and, while implemented by third party vendors, it is not prone to modifications or improvements by mitigation techniques that could be used in the SVM core.

To tackle this difference, a slightly different bitstream was used in fault injection where the communication infrastructure and the SVM core where placed in different regions of the FPGA allowing to study the contribution of these two parts to the overall DUT reliability. With this approach, faults could be injected in either part, separately, to analyze the individual contributions for DUT reliability, or in both parts simultaneously, for comparability with radiation results. It is worth noting that the AXI communication infrastructure was present in radiation experiments, being evidenced here only for the convenience of the analysis of fault injection results.

To further accelerate fault injection campaigns, part of the diagnostic logic was embedded in the ARM processor. The ARM application and FPGA bitstream were loaded from the Flash memory present at the ZedBoard development board. The board reset was implemented by power cycling using an automated power switch controlled by the fault injection campaign script running at the host computer.

Finally, an additional module was implemented in the FPGA to support fault injection. This module is based on Xilinx *Internal Configuration Access Port* (ICAP) and allows communication with the host computer that also runs the fault injection campaign script. This fault injection module, coded in VHDL, was not present in radiation experiments. This setup used for fault injection is depicted in Figure 4.1.

4.3.2 Assessment Metrics

In this work, bit-flips in the *Configuration Random-Access Memory* (CRAM) caused by radiation are referred to as fault. FF bit-flips are not possible as our design is fully combinational. Faults may halt the system execution or corrupt its outputs. When it halts the execution, it is defined herein as a *Crash Failure*. When it continues to output values, each output is classified using the same metrics as in Subsection 4.3.2.

Whenever a fault that has not crashed the system is detected, the entire dataset is run over the classifier. Each of the input samples score is logged for further classification into either *Critical Failure*, a *Tolerable Failure* or *Masked Fault*.

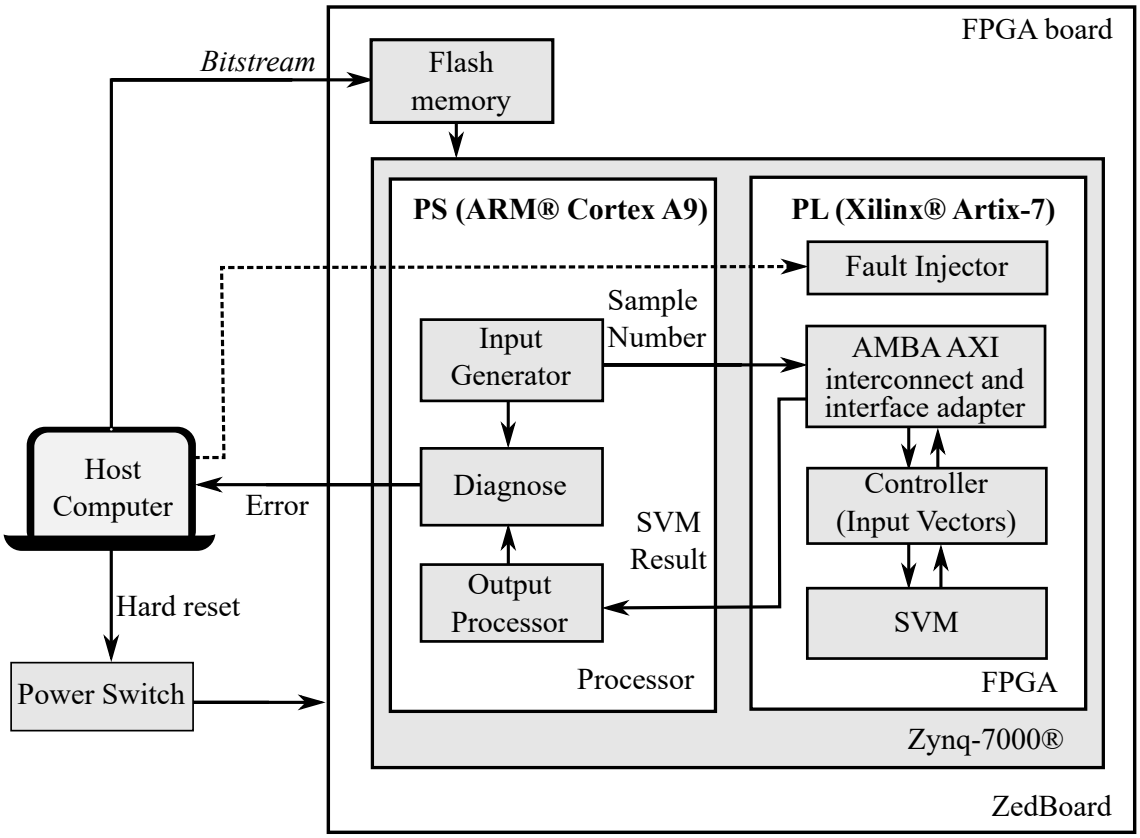


Figure 4.1: Zynq-7000 set-up under fault injection

4.3.3 Fault Injection Methodology

The total volatile memory available at Zynq-7000 device used in these experiments, including cache and RAM memory used by the processor system and the data and configuration memory used by the FPGA, amounts to approximately 32 mebibits.

The Xilinx Vivado design synthesis tool reports a total of 24.5 mebibits in the FPGA side for the device in use, of which approximately 4,9 mebibits (20%) corresponds to memory available to user data in the form of BRAM. Conversely, the remaining 19.6 mebibits (80%) is mostly CRAM that holds configuration for BRAM blocks, DSP blocks, CLB blocks and configuration for all possible signal routing throughout the FPGA switch boxes. The fault injection tool used in this work is targeted specifically to that 80% of FPGA memory relative to the CRAM memory.

In Xilinx 7 Series FPGAs the memory is organized in *frames* of 101 words of 32 bits. A frame is the minimal unit of CRAM memory that can be read or written to the FPGA using the Xilinx ICAP hardware block available at Xilinx 7 Series FPGAs. This is the same hardware block shared with the initial configuration of the FPGA or the par-

tial dynamic reconfiguration of the FPGA, but now used to read or write a single frame instead of loading or reading back the whole memory or a block of memory that holds a retargetable module.

The emulated fault injection can be implemented, therefore, by reading a single CRAM memory frame, changing its value and writing the frame back into the CRAM memory.

The injection flow used in this work [44] is presented in the sequence. The approach followed, named *random-accumulated*, consists in injecting faults in randomized positions in the CRAM memory region occupied by the DUT. After each fault injected the DUT is exercised with the completed set of input samples. The outputs are then logged according to 4.3.2. Those faults are accumulated in memory until a Crash failure is detected and then all faults are cleaned up by reprogramming the FPGA. This process is repeated until a significant number of events is collected. Then, a reliability curve is derived from the data. This curve is generated in order to visualize how the reliability falls when faults start to accumulate, i.e., the percentage of the dataset that was misclassified by the number of accumulated faults. This approach aims in emulating the accumulation of SEUs on the CRAM memory during the DUT operation, where one emulated fault injected would be equivalent to the number of particles given by the static cross section of the underlying device.

All the DUT modules targeted by fault injection were constrained to a rectangular CRAM memory region of 2,070 frames of 3,232 bits amounting to 6,690,240 bits. As an FPGA is a general-purpose device that is being programmed to a particular application, not all these CRAM bits are effectively used by the DUT as many CLB, BRAM, DSP blocks and most of the signal routing paths throughout the FPGA fabric remains unused. Those memory bits effectively required to program the FPGA to a particular application can be called *essential bits* [45]. In the case of the DUT in our experiments, the Xilinx Vivado synthesis tool reported a number of 1,059,559 essential bits in the case of the Binary SVM (Section 4.2.1) and 728,455 bits in the case of the Multiclass SVM (Section 4.2.2).

The diagnostic collected from DUT allowed the classification of each CRAM bit according to the criteria already defined in Section 4.3.2.

4.3.4 Results

During random-accumulated fault injection campaigns, the two DUT for Binary and Multiclass SVM were tested in three different physical floorplans on the FPGA, allowing faults to be injected separately on the SVM core, on the accessory communication modules and on all modules together.

Several fault injection campaigns were executed to explore the DUT reliability under accumulated faults amounting to more than 10^6 faults injected.

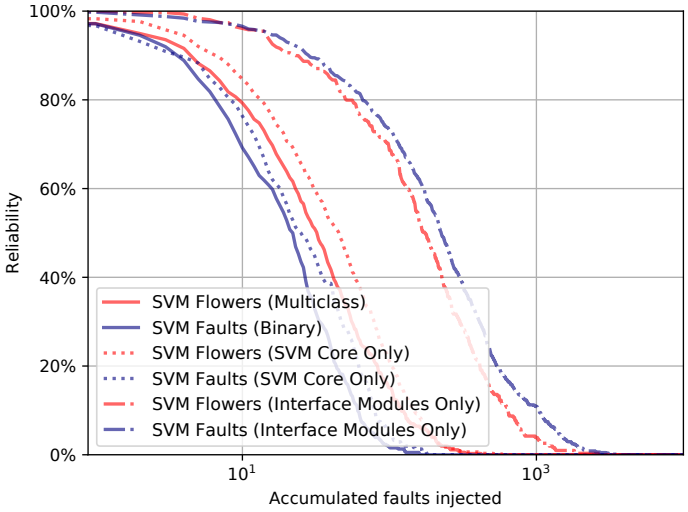
Although the rate of fault injection and the SEUs produced by thermal neutron irradiation shall differ at least by a factor relative to the device static cross section, the relative rank of the reliability curves is consistent among the experiments, as can be observed comparing Figure 4.2(a).

For the fault injection campaigns, whose results are presented on Figure 4.2(a), any failure, either tolerable or critical, were considered as a functional failure. Additional fault injection campaigns were executed with a relatively lax criteria where only critical failures, that is when the SVM produced an incorrect classification, was considered as a functional failure. These results are presented on Figure 4.2(b). This is a sound criterion when the SVM is used as a classifier because only the final SVM classification and its semantic meaning are relevant. It is worth noting that, in this implementation of SVM, the classification depends only on the signal bit of the numbers at the SVM primary output and not on the magnitude of those numbers.

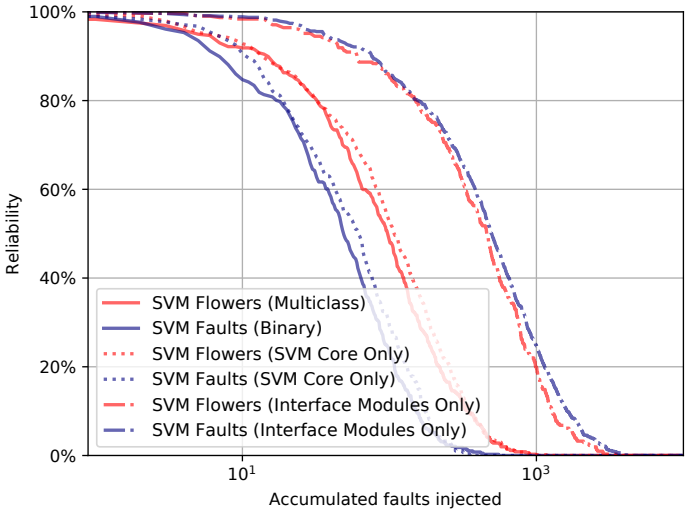
In Figure 4.2(a) and (b), it is noticeable that the Binary SVM is less reliable than the Multiclass version, as the reliability curve falls quicker. In [46], the authors have showed that the Binary SVM has a level of intrinsic fault tolerance. A more in-depth discussion on the reasons in presented later in Subsection 4.4.5. In our experiment, we have shown that the Multiclass SVM may be even more reliable. It is worth noting that more experiments should be conducted, as other factors may have played a role, such as the different datasets used.

Another aspect explored using fault injection was the impact of the microprocessor interface logic compared to the main SVM computation core. For this, fault injection campaigns were executed injecting faults only over the SVM core and only over the interface modules. The results are also presented on Figure 4.2(a) and 4.2(b), where we can observe that the overall DUT reliability is dominated by the SVM core with a sig-

nificantly higher reliability at the interface logic. This suggests that, despite the use of the third-party interface modules, there is still room for major improvements in the DUT reliability by implementing fault tolerance techniques to mitigate SEUs on the SVM core. It is to be noted that the BRAM was used by the interface modules in order to store the input samples. In a real-world situation, this would probably not be needed, as input samples would be generated by the environment, thus making the reliability numbers of the BRAM in our study not relevant.



(a) Fault injection, any failure



(b) Fault injection, critical failures

Figure 4.2: SVM Reliability.

4.4 Radiation Test Experiment and Results

This section describes the radiation experiments of both the Binary and Multiclass SVMs conducted with a thermal neutron source and an analysis of the obtained results.

4.4.1 Radiation Test Set-Up

The thermal neutron test has been conducted at the *Platform for Advanced Characterisation* (PAC-G), hosted by *Institut Laue-Languevin* (ILL), using the D50 thermal neutron source. Previous papers [43] have demonstrated the relevance of the usage of this equipment to perform reliability testing. It provides a thermal neutron beam with a spectrum ranging from few microelectronvolts to around 100 meV, with a peak around 13 meV. The captured flux (i.e. equivalent flux of 25 meV) is adjustable from 0 to 10^{10} $neutrons\ /\ (s \cdot cm^2)$. To keep parts of the board other than circuit, the board has been protected by a polyethylene sheet with a hole above the target chip. Each SVM architecture was tested individually with a constant flux of $6.94 \cdot 10^6$ $neutrons\ /\ (s \cdot cm^2)$. The Binary SVM was tested for 2 hours and 20 minutes, yielding a total fluence of $5.83 \cdot 10^{10}$ $neutrons\ /\ cm^2$, and the Multiclass SVM was tested for 3 hours and 8 minutes, giving a total fluence of $7.833 \cdot 10^{10}$ $neutrons\ /\ cm^2$.

In order to test both of the SVM architectures (Binary and Multiclass) designed, we have made use of the set-up is illustrated in Figure 3.7. The test vehicle used was a ZedBoard, that embeds a Zynq-7000 SoC. This is the same model of board and same part number used in our fault injection experiments, described in 4.3.1. On the PL part, two components have been instantiated: one of the SVM architectures, marked as SVM in the figure, and an indexed list of its respective input vectors, named the controller. The controller, when given an index, outputs to the SVM the input sample at that index. For example, when given as input the number 8, it will place on the instantiated SVM primary inputs the eighth input sample. The SVM module would contain either the Binary or the Multiclass SVM at a time, i.e. the Binary SVM and Multiclass SVM were tested separately. On the PS part, one module is responsible send to the controller the indexes while a second module reads the output of the SVM through an AXI interconnect and forwards it to a host PC through a serial port. The L2 cache of the ARM processor has been disabled to reduce the probability of faults affecting the PS [39]. No scrubbing

mechanism nor the Xilinx *Soft Error Mitigation* (SEM) core IP were instantiated. We are aware that these IPs would be very useful in a final implementation, as they would not let errors accumulate. However, these tools have a time delay in order to detect/correct the fault. This may still be not sufficient in a short term for our design, as it is fully combinational. Thus, these IPs were left off to observe a worst-case scenario.

4.4.2 Radiation Test Method

The radiation test methodology follows the one used in the work in Chapter 3, being the same for both the Binary and Multiclass SVM. One architecture was tested at a time. As for the previous, the set of input vectors is continually ran on the SVM instantiated in the FPGA. Whenever a fault is identified - by detecting a deviation in the output from the expected output - the set of input vectors is rerun twice on the platform, following by a reset and a reflash of the FPGA.

Figure 4.3: FPGA board installed at the D50 thermal neutron accelerator facility

4.4.3 Radiation Test Results for the Binary SVM

During the neutron radiation test campaign, we have identified 24 errors, of which 3 crashed the FPGA and 21 errors that allowed it to continue to produce results. No transient faults have been identified. Even though crashes have been responsible for 12.5% of the total number of errors, they are not related to the case-study SVM architecture design but due to either a fault in the device performing the serial communication with the control computer, i.e. the PS, or on the on the AXI module instantiated on the FPGA fabric, which can cause the PS to hang if it fails to perform a proper handshake. The obtained static cross-section and the FIT are respectively $4.11 \cdot 10^{-10}$ cm^2 and 2.67, considering New York's flux at sea level (6.5 $thermal\ neutrons\ /\ (h \cdot cm^2)$) [43].

4695 samples were processed by the Binary SVM with radiation-induced errors. Note that the total number of samples is not a direct product between the number of input sample and the number of faults, as it would be expected by the methodology described in Subsection 4.4.2. This is the case as in some cases, the FPGA would halt after a fault before reevaluating the complete set of input vectors. Of the total number of samples evaluated, 7% resulted in critical failures, while 21.4% have been tolerable failures and the majority, 71.6% were masked faults. From these results, it is noticeable that it is more likely for an error not to critically interfere with the application in this study case, as in 93% of the cases, the final classification of a sample would still be correct, recreating roughly the results in [46]. It is worth nothing that no fault mitigating or correction has been implemented on the Binary SVM, with the overall error resilience being an intrinsic characteristic of the algorithm.

4.4.4 Radiation Test Results for the Multiclass SVM

On the radiation campaign of the Multiclass SVM, a total of 16 faults were identified, of which 2 caused crashes. Again, the radiation induced crashes are out of the scope of this work . The obtained cross-section is $2.042 \cdot 10^{-10} cm^2$ with a FIT of 1.32 using New York thermal neutron flux at sea level.

A total of 2884 samples has been evaluated in the implemented Multiclass SVM, of which 2049 have been classified as masked fault, 799 as tolerable failure, and 36 as critical failure. As observed on the Binary SVM, the Multiclass SVM also has an intrinsic

tolerance to faults. The results indicate that only 1.2% of the evaluated samples on faulty Multiclass SVMs have been misclassified. Furthermore, only 3 out of 14 faults that had not crashed the FPGA have led to at least one critical failure.

4.4.5 Assessment of Results and Comparison of the SVM Architectures

First, we are going to discuss the effect of radiation-induced faults in Binary SVMs. On FPGA implementations of a Binary SVM, errors mathematically change the classification function (Equation 2.1) as they change either an x_i, an α or the calculation logic. The location of error may have a great importance on how it impacts the architecture. Errors on the least important bits of an x_i or α can potentially cause a small displacement of the classifier, making it less probable that a sample is misclassified. In terms of the architecture, the algorithm is a series of multiplications performed in parallel accumulated in a series of additions. This structure suggests that changes in the least significant bits are less likely to greatly impact the score of a sample, not causing a critical error. For example, an input sample that when evaluated on the SVM should output a score of 2.0, may have its score change to 1.9 if an error happened on the least significant bit, being still classified correctly. However, if an error happens on the most significant bits, the result could become -2.0, which would be a critical error as the signal of the score represents the final class. Note that different samples are affected differently in the event of a fault. Samples that have scores closer to zero are more likely to be misclassified, being sensible even to changes on not so important bits. For instance, a sample that has an original score of 0.2 is more easily made negative than a sample with score of 2. Thus, the distribution of the samples in regard to the classifier has a great impact on the intrinsic reliability of the algorithm. On the other hand, the training algorithm of the SVM maximizes the distance between samples of different classes, i.e making the score of samples as high as possible, corroborating to an augmented reliability, but still highly dependent of the dataset. Finally, the output of the case-study SVM is composed of 49 bits, of which 48 are irrelevant for the calculation, only the sign bit being used for the final result. This indicates a high level of fault tolerance even to faults close to the output. All these properties translated into a 93% level of tolerance to faults for the case-study Binary SVM.

As mentioned in Subsection 2.3.1, Multiclass SVMs are made of a composition of Binary SVM, thus inheriting the intrinsic reliability properties previously discussed. However, our Multiclass SVM presented even higher levels of fault-tolerance in comparison to purely Binary SVM, only having a 1% rate of critical errors, suggesting that it is more reliable than the Binary counterpart. This behavior has also been present in the fault-injection campaign, as shown in Figure 4.2(b), in which the reliability of the Binary falls more rapidly than the one of the Multiclass. One possible explanation is that parts of the Multiclass SVM are irrelevant when evaluating a sample. For example, in our case, there were three possible classes for an unknown sample: *virginica*, *versicolor* and *setosa*. As the Multiclass SVM for this study follows the One-vs-One approach, described in Subsection 2.3.1, three Binary SVMs have been trained, one for each pair of classes. Assuming that an unknown sample should be classified as *virginica*, the result of the Binary SVM to classify between *versicolor* and *setosa* is irrelevant as long as the output of the other two remain correct. Therefore, Multiclass SVMs may build an extra level of reliability when compared Binary SVMs, which is indicated by the radiation results. It is worth pointing out that the datasets used have been different, which may have an impact in the levels of reliability. Further evaluation using both fault injections and radiation tests would be needed to better compare the difference in terms of reliability between the two architectures.

4.4.6 Comparison with State-of-the-Art Works

Few authors have explored the radiation effects on machine learning algorithms as it is still a new field. The intrinsic fault tolerance of an FPGA implementation of ANN is evaluated in [26], in which the authors perform a fault injection campaign along with a heavy ion campaign. The work is complemented in [27], where the same architecture along with an FPGA implementation of a CNN, a very popular variant of ANN for image processing applications, have been evaluated under the effect of neutrons. GPU implementations of CNNs have also been evaluated under neutron radiation in [28].

In [26, 27], the authors have made used of the same dataset that we have used for our Multiclass SVM to train an ANN. Also, they have used the same FPGA platform. When comparing with their results, the authors have found that 65% of the faults led only to tolerable failures. In our case, we achieved 79% in respect to that, suggesting that

SVMs may have a higher reliability in comparison to ANNs. We have also had reliability figures comparable to those of [28], even though the datasets used are different, but still with multiple output classes. Using GPU and CNNs, the authors have found that around 82% of the faults in one configuration bit of their GPUs would lead to no critical error. When comparing to GPU implementations of CNNs, while GPUs present better fault tolerance, FPGA implementations cannot be ruled out as they may be faster for some applications [41].

4.5 Conclusions

This work presents the first evaluation of SVMs under thermal neutron radiation along with the first assessment of radiation effects on Multiclass SVMs. Furthermore, both architectures were also thoroughly evaluated with an extensive fault injection campaign to correlate with the radiation results. On both the radiation and fault injection campaigns, the Multiclass SVM presented an overall higher reliability when compared to a Binary SVM. It is worth noting that neither designs had any error detection nor error correction mechanisms implemented, suggesting that the Multiclass SVM has a higher intrinsic reliability. Also, in our experiment, the Multiclass SVM performed better in terms of reliability than an ANN trained for the same dataset, suggesting that it may be more reliable.

Acknowledgments

This work has been partially supported by LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01), funded by the French government program "Investissement d'avenir"; and MultiRad project funded by Région Auvergne-Rhône-Alpes's international ambition pack. In addition, we would like to thank R. A. Guazzelli (TIMA), Alexandre Coelho (UFC), and R. V. Della Giustina (ILL) for the help with the radiation test logistics.

5

Assessment of Machine Learning Algorithms for Near-Sensor Computing Through Fault Emulation

Contents

5.1 Introduction

Researchers have started to investigate the impact of radiation-induced soft errors on the reliability of ML techniques. For instance, the authors in [47] [48] examine effects of soft errors in CNNs. In [26] [27], 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. [49] investigate how the presence of soft errors in GPUs can reduce the reliability of a CNN. Rosa et al. [50] 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 [51] have proposed a framework that performs fault injection at specific *Deep Neural Network* (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 [29] has conducted soft error resilience analysis of Bayesian machines, the work in [42] has focused on a binary support vector machine implemented in an FPGA.

Different from the above works, this Chapter assesses and compares the reliability of two ML algorithms – feed-forward ANN and 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.

5.2 Case-Study ML Algorithm Models

In this work, 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. A more in-depth description of the models is presented in Chapter 2.

5.3 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 *GNU 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 fault emulator is further described in Chapter 7.

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 5.1): (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.

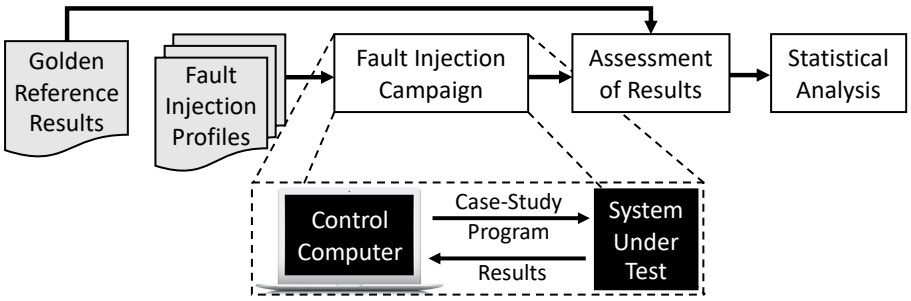


Figure 5.1: Fault injection-based method for assessing the reliability of an ML algorithm (case-study program) running on a low-power processor (SUT subcircuit) under effects of single soft errors.

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.

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)
```

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.

Assessment of Results

In order to assess the algorithms, the output (or lack of output) of the implementation is compared with a golden reference. This output is then assigned a label as described in

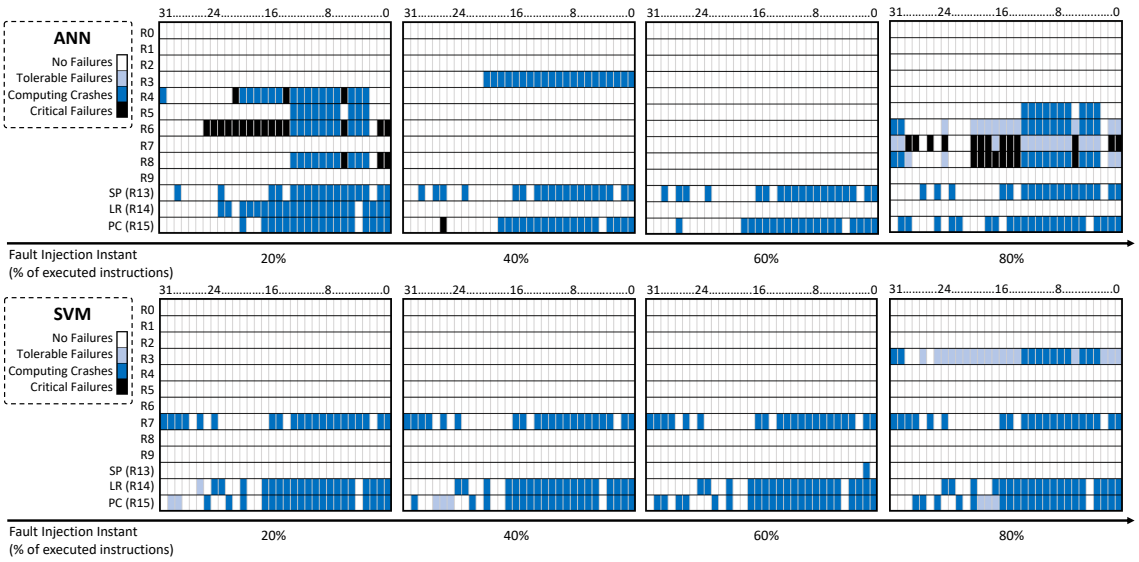


Figure 5.2: Snapshots of the fault injection instants in the processor register map for ANN (top) and SVM (bottom) algorithms using the same set of input vectors. This shows which registers are further stimulated and therefore the most susceptible to radiation-induced soft errors.

Section 4.3.2.

Statistical Analysis of Results

Equation 5.1 below analyzes the ratio of the total number of *Critical Failures* to the total number of fault injection profiles assessed according to 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.

$$\%CriticalFailures = \frac{\#CriticalFailures \times 100}{\#FaultInjectionProfiles} \tag{5.1}$$

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 [52]. 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.

5.4 Fault Injection-Based Assessment

This section analyzes results of experiments that have been carried out applying the method described in Section 5.3 for comparing the case-study ML algorithms defined in Section 5.2.

5.4.1 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 [11] 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 5.3, 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.

5.4.2 Analysis of Results by Register

Figure 5.2 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 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 5.2 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 5.2. In contrast, *SP*

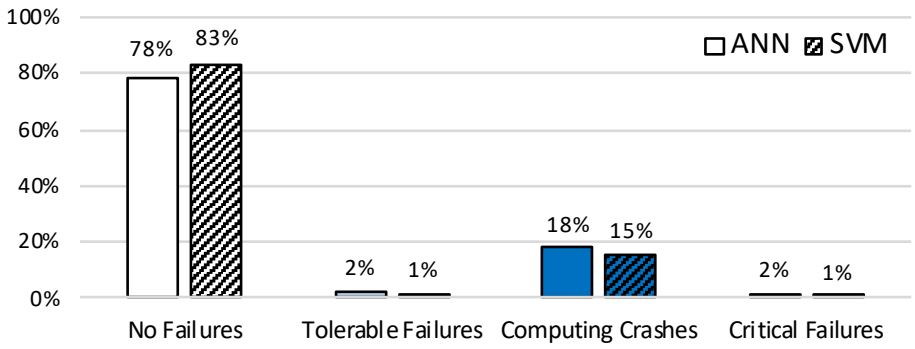


Figure 5.3: Summary of situations induced by the fault injection campaign in the ANN and SVM algorithms.

is more sensitive to ANN than SVM, as our implementation makes heavy use of the stack.

Analysis of Global Results

Figure 5.3 presents the summary of the fault injection campaigns for the ANN and SVM algorithms. The values have been calculated as shown in section 5.3, combining all results.

Looking closely in Figure 5.3, *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 5.3 suggests that the SVM algorithm presents a slightly better reliability than the ANN when under the influence of single soft errors.

5.5 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 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.

5.6 Conclusions

This work 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.

6

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

Contents

6.1 Introduction

In this Chapter, the assessment of the effects of radiation-induced faults on low-power implementations of SVM and ANN conducted in Chapter 5 is expanded. The implementations of both models were evaluated through neutron radiation tests conducted with a neutron generator. Furthermore, a third algorithm was also evaluated: the RF. Thus, in this work, three traditional ML models from three distinct families, i.e. models with different behaviors, selected based on their popularity and applicability – feed-forward ANN from family of neural networks, SVM from statistical learning and RF from the group of decision trees – running on a low-power are assessed and compared.

6.2 Case-Study Machine Learning Models

This section briefly describes three prominent ML models that have been studied in this work: ANN, SVM and RF. These 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. The models are described in Chapter 2

6.3 Implemented Case-Study ML Algorithms

This section describes the ML models implementations used for this work along with the dataset that has been used to train them. We have used the STM32 Nucleo-L45RE-P board as target, which is an option for Endpoint-AI applications. We refer to the board as SUT onwards. The implementations were designed to output the mathematical values calculated by the ML models, herein defined as score. For this work, we have used the Iris flower dataset [11] to train and evaluate the algorithms.

ANN Implementation

To generate our ANN model, we have used the Keras Machine Learning framework. For the ANN structure, we have opted for and ANN with 1 hidden layer with 8 neurons along

with the input and output layers, with 4 and 3 neurons, respectively, as it was enough to achieve good precision on our dataset. The generated model was extracted and coded in C to execute in our target platform. The model is composed of a 67 weights. The output of the implemented ANN model is the output of the three neurons in the output layer. Our implementation uses 1.53 kB of RAM and 0.09 kB of the Flash on our SUT.

SVM Implementation:

The SVM model was trained using the MATLAB implementation of the SVM training algorithm using our training dataset. As we have used the One-vs-One model, the training algorithm generates three binary SVM models for Iris dataset, one to distinguish between Setosa and Virginica, one between Virginica and Versicolor and one between Setosa and Versicolor. The generated models, i.e. Support Vectors, αs and bias, were extracted and coded in C to execute in our target platform. The Setosa x Virginina and the Setosa x Versicolor models are composed of 2 pairs of SVs and αs, while the Virginica x Versicolor model is compose of 16 pairs of Svs and αs. The output of our implementation is the numerical results of each binary SVM. This is done to allow for better observability of deviations in the expected behavior. In terms of memory usage, 2.13 kB of RAM and 13.15 kB of Flash storage were used.

RF Implementation:

For RF implementation, we trained our model using Scikit-Learn. For the RF structure, we have used 10 BDTs. After the training, the obtained BDTs had the following number of nodes: 3, 5, 5, 10, 4, 8, 7, 5, 7, 7. As for the other model, the weights and structure of the BDTs were extracted and coded in C. The output of the final implementation is the output of the 10 BDTs. The memory usage of the implementation was 18.03 kB of RAM and 0.09 kB of Flash.

6.4 Radiation Test Assessment

6.4.1 Radiation Test Set-Up

The case-study ML models have been tested using the GENEPi2 neutron acceleration facility at Laboratory of Subatomic Physics & Cosmology (LPSC), Grenoble. The equipment generates a 14 MeV neutron beam with a flux with a maximum flux greater than the 14 MeV neutron flux at 40,000 ft by a factor of 10^{10}. The SUT is placed directly in front of the accelerator at a distance of 5cm and then the flux is calibrated remotely. The SUT is connected to a control computer outside the radioactive chamber through an USB cable. The communication between the SUT and the control computer was protected with a checksum. A schematic of the set-up is shown in Figure 6.1. Figure 6.2 shows a picture of the set-up assembled in the facility.

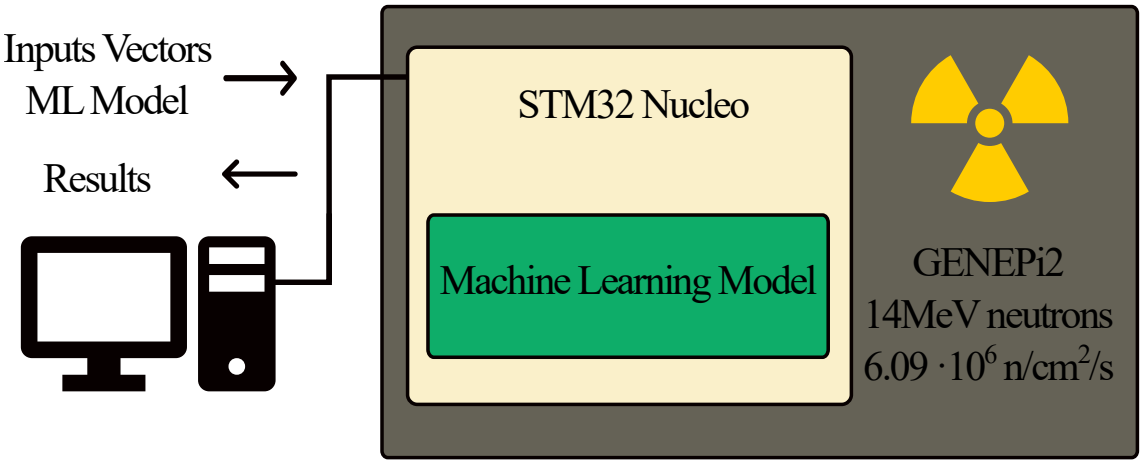


Figure 6.1: Schematic of the test setup for the radiation campaign.

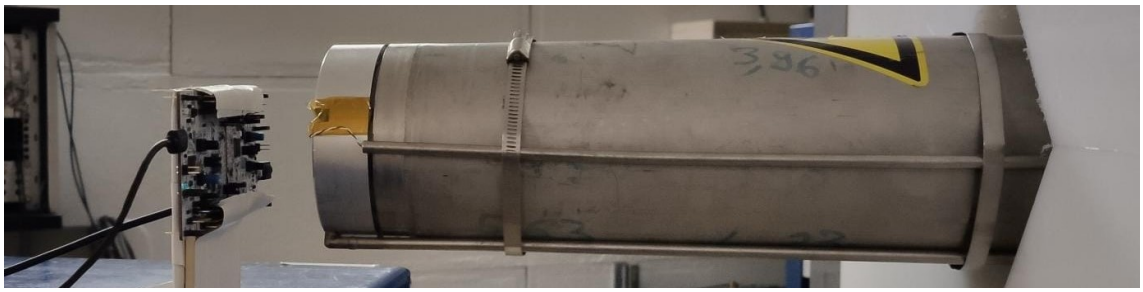


Figure 6.2: SUT mounted in the GENEPi2 14 MeV neutron source.

As mentioned in Section 6.3, each ML model implemented outputs a score. Due to radiation induced effects, the score output may deviate from the golden reference. To assess

the effect of radiation induced faults to evaluate an ML implementation effectiveness, a score output from SUT is classified using the metrics defined in Subsection 3.3.2.

Herein we define faults as radiation induced events in the SUT, also referred to as SEUs. A failure is a consequence of a fault, observed analyzing the primary output of the system.

Each ML model is loaded in the SUT at a time by the the control computer. The control computer then sends one input vector at time to the SUT and waits for the output. Once it receives the results, the control computer compares the result to a golden reference and classifies it following the metrics in Subsection 6.4.1. If a tolerable or critical failure is detected, it is certain that a fault has occurred in the SUT. Then, the complete set of input vectors is reevaluated in the SUT and logged to verify if the fault affects future executions of the algorithms. Only the results obtained when a failure is present are accounted. When this process is finished, the board is erased and reprogrammed to prevent the accumulation of errors. The SUT is also reprogrammed if a communication error is detected.

6.4.2 Radiation Test Results

The SUT has been irradiated for a total of 14 hours and 16 minutes, yielding a total fluence of $3.79 \cdot 10^{11}\ n/cm^2$ (average flux of $6.9 \cdot 10^{6}\ n/cm^2/s$) distributed among the ML models accordingly. The distribution of campaign time and fluence as well as the number of faults and the cross-section of each implementation is presented in Table 6.1. Figure 6.3 presents the percentages of the types of failures and the absolute numbers of each. A more detailed map of how some of the faults identified affect the effectiveness of the algorithms is show in Figure 6.4.

Table 6.1: Distribution of the irradiation time among the ML models and the identified failures

ML model	Radiation Time	14 MeV Neutron Fluence	Faults	Cross-section
ANN	4h53min	$1.21 \cdot 10^{11}\ n/cm^2$	10	$8.23 \cdot 10^{-11} cm^2$
SVM	3h54min	$9.81 \cdot 10^{10}\ n/cm^2$	7	$7.13 \cdot 10^{-11} cm^2$
RF	5h26min	$1.5 \cdot 10^{11}\ n/cm^2$	6	$4.43 \cdot 10^{-11} cm^2$

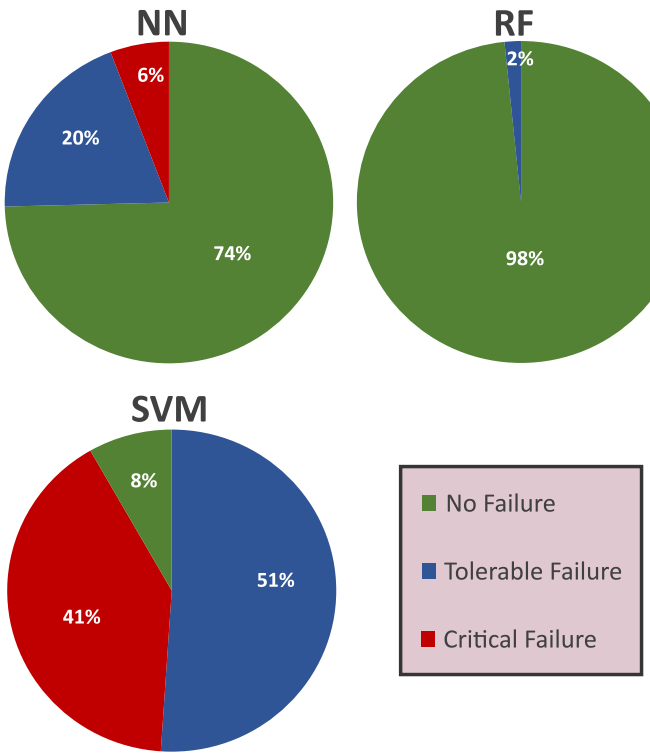


Figure 6.3: Results obtained during the radiation test logged according to Subsection 6.4.1.

As per Figure 6.3, the three ML models have not presented critical failures in the majority of the cases, with the SVM having the largest percentage of critical failures at 24%, ANN having 6% and no critical failure being observed on our RF implementation. The results indicate that, for the three ML learning, in the majority of the cases, the radiation induced faults do not cause deviations important enough to result in misclassification. It is worth pointing out that no protection mechanics was implemented, suggesting that they retain a certain level effectiveness under radiation intrinsically.

Figure 6.4 shows that faults affect more than one input vector for ANN. One possible cause for this are radiation induced faults on weights stored in the memory of our implementation. As mentioned in Section 6.2, the ANN is composed by a layered set of neurons, each neuron performing a weighted addition of the results of the neurons just before it. Thus, changes on weights are one of the possible causes for deviations on the score. In addition to it, internally, the neurons are implemented as loops. Faults on the memory positions or registers storing loop bounds may cause loops to stop early and output unpredictable results. Both weights and loop bounds reside in the memory of our SUT, being loaded onto registers whenever they are needed. If a particle attacks these

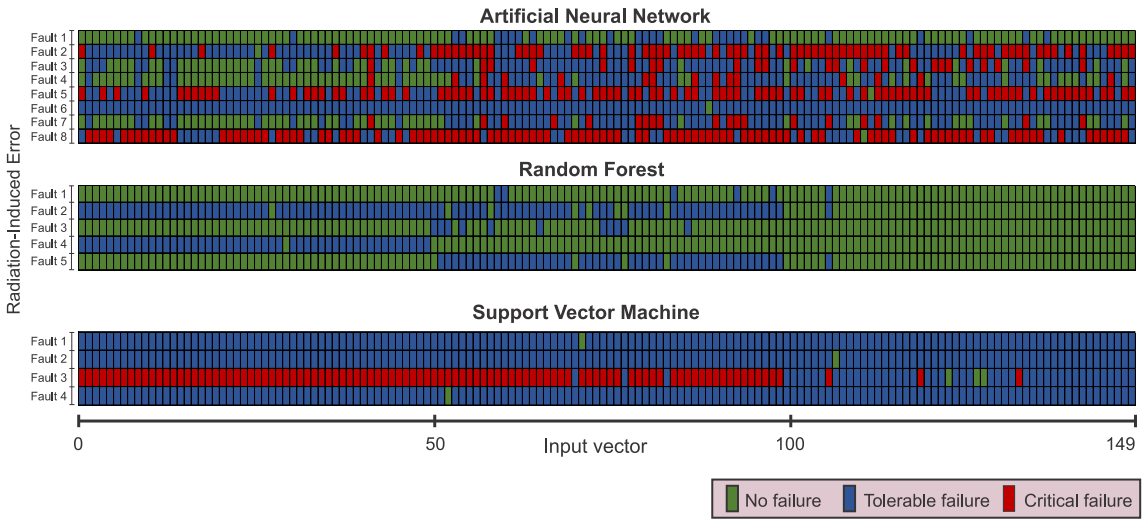


Figure 6.4: Effect of some of the identified faults on the in input dataset. Each line represents an identified fault and the columns represent input vectors of the input dataset. For instance, The top left square indicates that the input vector 0 output was classified as no failure when a fault was identified on the ANN.

values while they are on memory (and given that these values are assumed as constants in our implementation), it is very likely that this fault may generate multiple failures, as every subsequent input vector will be evaluated with now incorrect weight/loop bounds. On the other hand, if a particle affects these values while they are on registers, it may only affect the current input vector being evaluated, as the incorrect value will not propagate to memory, thus not affecting future calculations, but this behavior, even though possible, was not observed in our experiment.

On the SVM, Figure 6.4 indicates that faults have a tendency to provoke tolerable failures. Mathematically, the SVM is close to the ANN, being constituted mainly of a MAC operations. Thus, we expected that the profile of the generated failures would have been similar, but the rate of tolerable failures on the SVM was far superior than the ANN. We will investigate the reason behind it in with further fault injection and radiation campaigns. Also, it is still unclear why Fault 3 in Figure 6.4 had an elevated number of critical failures, which we plan to evaluate further.

RF was the only ML model in which no critical failures were observed, as shown in Figures 6.3 and 6.4. One of the possible reason for this behavior is that the RF model relies on the independent output of 10 substructures (10 BDTs) to perform a classification, while the ANN and the SVM results are not totally independent of each other. As these 10 substructures are independent and only vote for the final class, a fault on one may

cause one of the votes to be wrong, while the others maintain the final classification correct. Furthermore, the number of tolerable errors for the RF is also lower than the SVM and ANN. One hypothesis is that the ANN and the SVM directly output the results of mathematical calculations, while the RF outputs the constant values stored in the leaves of its BDTs. As weights only control how the tree is traversed in the RF, if a radiation induced fault in a weight does not change the path followed when evaluating a sample or if affects a node that is not used for a specific input vector, the fault will not generate a failure. Besides that, for the SVM and the ANN, a faulty weight will effectively be used on the classifier mathematical calculation and is more likely to propagate to the output, i.e. failure. Finally, our RF implementation made heavy use of dynamic allocation. Hence, we plan to investigate if this could have been one of the reasons it did not present critical failures in our experiment.

6.5 Conclusions

This Chapter provides evidences of the effectiveness of three ML implementations (ANN, SVM, and RF) under neutron radiation effects. All the implementations have presented a certain level of effectiveness under radiation effect even without protection mechanisms. Furthermore, the RF model presented the overall best effectiveness, with no critical failures being identified, which may be a consequence of how the model is intrinsically constructed, but further evaluation is necessary.

7

Development of a Fault Emulation Tool

Contents

7.1 Introduction

As mentioned in Chapter 1, fault injection is a viable way to analyse how a system may behaves when exposed to the effects of radiation. In this Chapter, a tool for fault emulation, i.e. injecting faults on the physical device itself, developed under the scope of this thesis is presented. It uses a popular debugging tool as back-end, the GDB, allowing the fault emulator to be ported to any platform that supports it. As proof of concept, the fault emulator was used in two different platforms: an STM32 Nucleo microcontroller and a Raspberry Pi. On the STM32 target, the implementations of 3 ML algorithms were tested. The results of this experiment are show in Chapter 5. In the Raspberry Pi, an implementation of the *Novel Quartenion Kalman Filter* (NQKF) algorithm was conducted. While the NQKF algorithm is not in the scope of the thesis, this experiment allowed for further validation of the developed fault emulator.

7.2 State-of-the-Art

Fault emulation is highly dependent of the target platform. For instance, the strategies for emulating faults on an FPGA are vastly different than the ones for emulating on a microcontroller or a GPU. On FPGAs, there are two major approaches: intrusive and non-intrusive. Examples of intrusive are works FITO [53] and NETFI [54], the latter having being used in Chapter 3. The idea on intrusive techniques is to modify the netlist of the target system to allow for faults to be emulated. In the FITO tool, a list of faults to be injected is provided to the fault emulator. Then, for each fault in the list, the original synthesizable code is modified as to contain intended fault. The newly generated code is then emulated on an FPGA. On the other hand, NETFI modifies the synthesis libraries in order to add extra logic gates on the system which allow for the emulation of bitflips. These logic gates are then externalized as primary inputs of the system to be piloted by

a soft-core processor instantiated in the FPGA fabric. These techniques are useful to estimate the failure rate of ASIC before it is sent to production, as the modifications in the netlist conducted reflect the effects of radiation on physical parts of the circuit at a gate level. On the other hand, non-intrusive techniques rely on changing the configuration memory of the FPGA. An example is the technique used in [26] and in Chapter 4. The tool uses partial reconfiguration to flip one bit of the configuration memory at a time, simulating one of the effects radiation has on memory-based FPGAs. Different than the intrusive technique, this allows for an estimation of the failure rate when the FPGA is the final platform intended for system. Furthermore, for ASIC development, tools like Cadence Incisive and Mentor ModelSim allow for fault injections at an *Register-Transfer Level* (RTL) level without emulation.

Emulating faults on GPUs and microcontrollers bring other challenges. If the RTL for the system is available, it wpuild be possible to emulate the architecture on an FPGA and use the aforementioned techniques. However, the RTL of commercial versions of such products often is not available. For these cases, debug tools are helpful allies. In [55], the authors make use of the debug tools specific for the their target platform as a mechanism for fault emulation. In [56], the authors use the GDB debug tool to manipulate the register files of GPUs. In this work, similar to [56], a fault emulation tool is built based on GDB, but instead of targeting GPUs, the main target are microcontrollers and processors.

7.3 Implemented Solution

The fault emulation suite was implemented using GDB, an open source debugger, as back-end, thus integrating with any platform that has support to it. In general, a debugger is used to control the execution of a target program with much more observability than when ran natively. The fault emulator is implemented as a Python library that provides a level of abstraction to an underlying child GDB project. At its initialization, the fault emulator spawns a child GDB process and controls it to inject the faults. The debugger has some key features that allow for the implementation of a fault emulator, notably breakpoints, stepping, and the read and write access to memory position and registers, which have been abstracted as function calls in the fault emulator.

Breakpoints are used to interrupt the execution at specific line of codes. Often, the

user wants to explore the effect of faults in specific points of the execution. By using breakpoints, he/she can select in which points to interrupt the execution in order to inject a fault. The fault emulator library provides a function to allow the user to straightforwardly set breakpoints. When the execution is interrupted, the injection of the fault is done by performing writing operation to either the memory or an specific register, which is supported by GDB and abstracted by the fault emulation library. In order to have a better control of how many bits are being changed, the fault emulator also provides a read function, also for memory and registers. With this function, the user is able to read the current value of the register he/she wants to affect, change as many bits as desired and write back the result.

Being implemented as library grants flexibility to the user. The order in which faults are going to be injected as well as the number of bits and positions is completely controllable. Also, if the target program needs to communicate with other programs to, for instance, receive inputs, this can be integrated with the fault emulation control in the same script. In terms of logging the results, as the needs vary depending on the target program, it is left to the user to write its own, but this is done using standard Python.

7.4 Proofs of Concept

The fault emulation library was used in two platforms. They are described in the sequence.

7.4.1 STM32 Nucleo Target

The first work conducted had as target a STM32 Nucleo-L452RE-P development board, which comprises an Arm Cortex M4 processor. The goal was to evaluate the effect of faults on three ML algorithms executing on the board. Porting the fault emulator to this platform was possible as the GDB is supported by the Nucleo framework.

The overall fault emulation architecture for the STM32 is presented in Figure 7.1. The Nucleo board is very limited in terms of resources, not being able to run GDB nor the fault emulator embedded on it. The GDB support for the board is done through a GDB server running on a host computer that interacts with a JTAG probe embedded in

the board. When a GDB server is open, it waits for a connection of client GDB process via the network. Debugging commands issued on the client are forwarded to the server, which will manipulate the board to perform the command. The original implementation described in Section 7.3 underwent small modifications to allow for its child GDB process to access the GDB server.

To validate the implementation, the fault injector was used to analyse the reliability of three ML algorithms. The results are presented in Chapter 5.

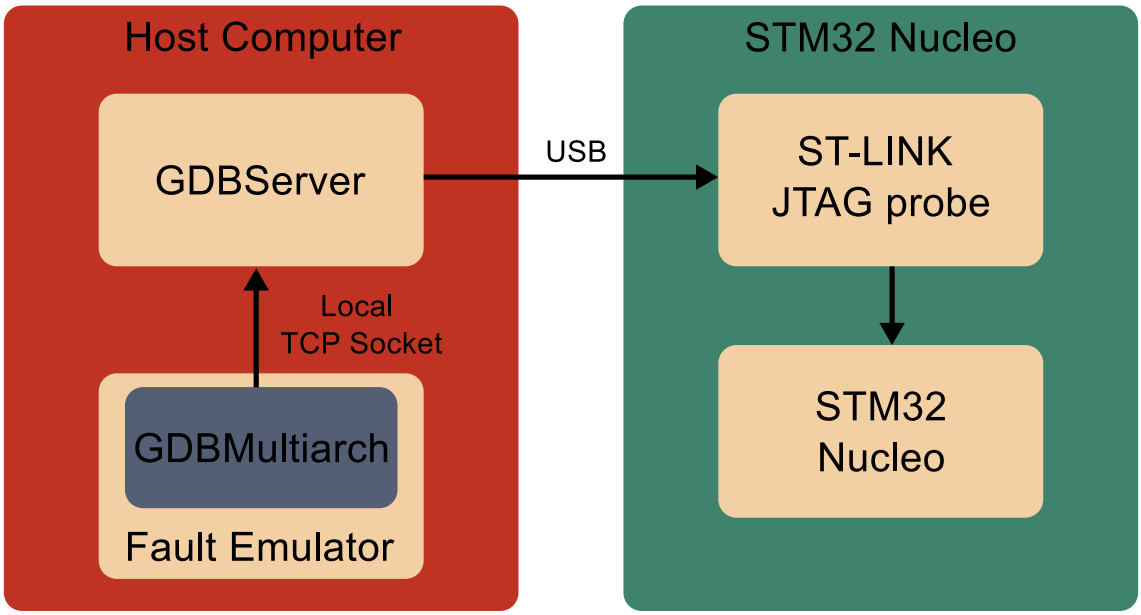


Figure 7.1: Fault emulator architecture.

7.4.2 Raspberry Pi Target

To further explore the possibilities opened by the fault emulator, it has also been ported to a Raspberry Pi 4B board, which embeds Quad core Cortex-A72 (ARM v8). Different than the use-case described in Subsection 7.4.1, the fault emulation on the Raspberry Pi 4B is self-contained, i.e. there is not a need for a control computer. As the board is capable of running full-fledged operating systems, it is possible to run GDB embedded on it, eliminating the need for a host computer.

Target algorithm

The fault emulator was used to test reliability of an implementation of the NQKF algorithm. The NQKF is an attitude estimation algorithm, a common requirement for

nanosatellites and *Unmanned Aerial Vehicles* (UAVs). Normally, such applications contain a variety of sensors, the most common types being gyroscopes, magnetometers and accelerometers. The NQKF analyses the data from these sensors and estimates the attitude, i.e. the orientation, of the physical system. On UAVs, for instance, this is particularly useful for the implementation of stabilization algorithms.

The algorithm uses quaternions to represent the attitude of the physical systems. This approach provides advantages over the more intuitive Euler angles (ϕ, θ, ψ, or roll, pitch and yaw), as the latter is subject to singularities, i.e. points where the attitude is not defined, and gimbal locks, i.e. a loss of a degree of freedom in the representation due to how the attitude is currently being represented. The NQKF stores the current quaternion representation of the system. It then samples the sensors and updates it accordingly. The inner details of the NQKF algorithm were left out as they are out of the scope this thesis.

Fault Emulation campaign

The NQKF was implemented in C++ and compiled for the Raspberry Pi 4B. In order to test it, a test case was generated. A set of 10 subsequent artificial samples of gyroscope, magnetometer and accelerometer were generated. The fault injection campaign was then conducted in the following steps: (1) *Generation of Golden Reference*; (2) *Generation of Fault Injection Profiles*; (3) *Fault Injection Campaign*; (4) *Evaluation of Results*.

1) Generation of Golden Reference: First, the NQKF is executed using the artificially generated samples in order to obtain the expected output of the algorithm., i.e. the golden reference.

2) Generation of Fault Injection Points: A set of fault injection points is generated. These faults are used by the fault emulator to pilot the fault emulation campaign. The fault points generated are described in Algorithm 2. For each register in the processor, random execution points in the algorithm timeline are chosen. Then, for each bit of the register (one at a time) and for each at each of the 10 artificially generated samples, a fault is injected. Note that, at most one bitflip is injected at a time.

3) Fault Injection Campaign: Each fault injection point is emulated in the SUT, one at a time. The NQKF implementation is executed until the random breakpoint. Then, a bitflip is forced in the bit position indicated. The program execution is resumed and the output is logged.

Algorithm 2 Generation of fault injection points

```
1: for x in [set of processor registers] do
2:     for y in [small sample of random breakpoints] do
3:         for z in [set of processor register bit] do
4:             for i in [range from 0 to 9] do
5:                 FaultInjection(x, y, z, samples[i])
```

4) Evaluation of Results: For each fault injected, the algorithm is run to completion. The output is stored and classified as follows:

- Match: The output matches the golden reference.

- Mismatch: At least one bit of output differs from the golden reference.

- Crash: The program halts or returns invalid data.

Fault Emulation Results

A total of 48,000 faults were injected during the campaign. Of these, 46,410 (96.69%) caused no perturbation in the system, i.e. the output of the system matched the golden reference even with a fault being present (**Match**). **Crashes** were the most common type of failures, amounting to 1500 (3.12%). **Mismatches** were the least common outcome, with a total of 92 (0.19%) having been identified.

The distribution of the results per register is shown in Figure 7.2. Failures (**Crashes** or **Mismatches**) were only on observed on the registers *sp* and *r11*, the latter also referred to as *fp*. In terms of the **Crashes**, they were all caused due to illegal memory access. A modern *Operating System* (OS) implements memory access mechanism. Whenever a process tries to access a memory region that has not been previously allocated, the OS intervenes by sending a SIGSEV signal to the process. If this signal is not captured and treated, it causes the process to crash with the *Segmentation Fault* error message. The *sp* and *r11* registers control the access to the process stack, which is used to store variables and implement function calls. If fault happens on them, it is possible that the process will not behave correctly anymore. For instance, the address stored may now point to an invalid address. Whenever it is used, i.e. whenever the stack is accessed, it will cause a memory violation. With a small change in the value, the register may still contain a valid address, but the whole workflow of program could be compromised. For instance, if the return address is expected to be at $sp-4$ and a variable is on $sp-8$ and if sp is deviated by 4 due to a fault, the program may access the variable memory position and interpret it

as the return address of the function. This could cause the program to jump to an invalid memory position, causing a memory violation.

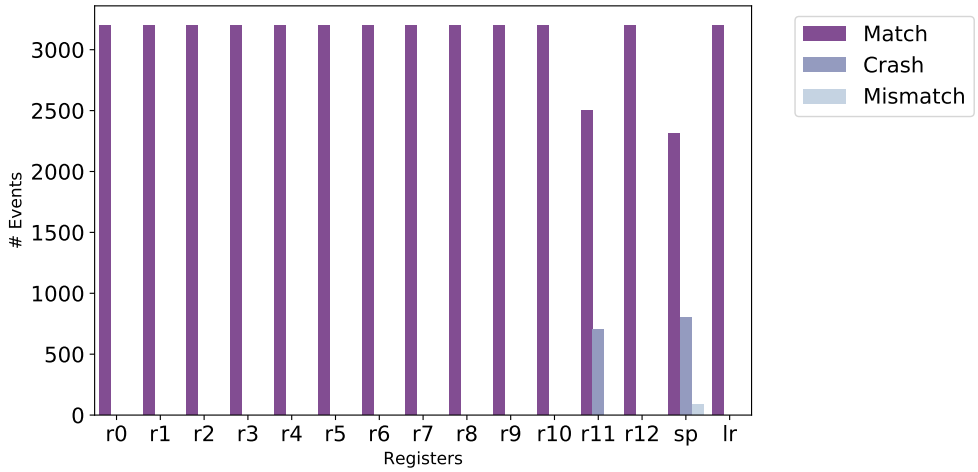


Figure 7.2: Results of the Raspberry Pi fault injection campaign organised by register.

Mismatches were also observed when faults were injected on the *sp*. As mentioned, the *sp* is used to access the process stack, which store the value of variables. A fault *sp* may cause it advance one memory position. Supposing that a variable A is located normally at $sp - 8$ and that a variable B is on $sp - 12$, if the *sp* advanced one position due to a fault, the content of the variable A may be read when the program actually intended to fetch B. If this is a value that is used as part of the calculation of the NQKF algorithm, it may affect the output, leading to a **Mismatch**. Further investigation is still needed to prove this hypothesis.

Moreover, no failures were detected when injecting faults on general purpose registers, even though they are possible. A general purpose register may, for instance, contain input data, intermediate results and even the output of the system. Affecting this data could lead to a **Mismatch**, but that was not observed in the experiment. Furthermore, these registers may contain array indexes or memory pointers. A fault in these cases could lead to either an access of invalid position, leading to a **Crash**, or the access to a valid but yet not intended memory position, potentially leading to a **Mismatch**. A possible reason for not observing such failures is that the implementation itself contains a large number of variables and pointers, more than the number of general purpose registers. This may cause the program to constantly back-up the registers values on the stack. If a fault is injected on register whose value has been backed up to the stack, the fault may be masked

as the program may fetch the data from the stack before using it. Further examination of the results is necessary to confirm this hypothesis.

7.5 Conclusions

In this Chapter, a fault emulator developed in the context of this thesis is presented. The tool was developed using GDB, an open-source debugger, as back-end. Due to the popularity of the GDB, various platforms provide support to it, which grants great flexibility to the fault emulator. In this work, the fault emulator was used in two different platforms with vastly different configurations, a Nucleo STM32 and a Raspberry Pi. In the Nucleo STM32, implementations of ML algorithms were tested, with the results being presented in Chapter 5. In the Raspberry Pi target, an implementation of a NQKF was tested. While the NQKF algorithm is out of the scope of the thesis, this experiment was useful to validate the tool. In this test, most of the faults injected in the NQKF implementation did not lead to failure, with registers *sp* and *r11* being identified as the most sensitive.

8

Conclusions

The general public often questions the amount governments spend on space programs, but space exploration, among innumerous other benefits, uncovered mechanisms that have threatened every electronic computing system built: radiation induced faults. Today, every commercial plane that has automatic guidance system, i.e. autopilot, has to account for such effects, as not doing so would risk an uncountable number lives. Furthermore, due to the unprecedented success of ML, it is important to assess how they fare against this effects. In fact, experiments have been done on using ML algorithms for self driven cars and, at the time of this these, there is ML algorithm on the multibillion dollar project on the surface of Mars. A failure on the first example could cost the loss of human life, while on the second one, it could harm an expensive mission. In this thesis, the reliability of multiple ML was assessed through both accelerated testing, i.e. radiation campaigns, and fault emulation. In this chapter, the results and contributions obtained during this work are summarized are discussed. A final discussion on the perspectives for the future is also presented to close the discussions of this thesis.

8.1 Contributions to FPGA implementation of Machine Learning

In the first two chapters of this thesis, the reliability of 2 FPGA implementations were evaluated. In the Chapter 3, the evaluation of the reliability of a binary SVM under the effect of radiation induced faults is presented, being the first work in the literature where an implementation of a SVM was irradiated. In this experiment, a custom implementation of the algorithm was first evaluated through a fault emulation campaign, which showed that the implemented model contained an intrinsic level of fault tolerance, meaning that even without any protection mechanism, it was capable to mitigate part of faults. To complement the fault emulation study, a irradiation campaign was conducted. In this campaign, it was also observed that the implementation had an intrinsic level of fault tolerance. Compared to the state of the art, the preliminary results of this first experiment showed that the SVM implementation had a marginal advantage in terms of reliability to other implementations of ML algorithms

Following this work, in Chapter 4, the work on the previous chapter is expanded by performing similar experiments on a multiclass SVM and on the same binary SVM FPGA implementations in order compare the reliability of both. This was the first work in the literature to evaluate the the reliability of a multiclass SVM under radiation effects and the first to test SVM implementations under the effects of thermal neutrons. First, extensive fault emulation campaigns were performed on both implementations. This showed that that the multiclass SVM had an advantage over the binary SVM in terms of intrisic reliability, i.e. reliability when no protection mechanism is implemented. After the fault emulation campaigns, both implementations were irradiated using a thermal neutron source, in which the result corroborated what was observed in the fault emulation campaigns, as the multiclass SVM also exhibited better reliability when compared to the binary SVM. A possible reason for this behaviour is that the multiclass SVM is composed of multiple binary SVMs, each trained for a subset of the training dataset. This intrinsically creates some redundancy. For instance, for the case-study in this work, the multiclass SVM was composed of three binary SVMs. A correct classification relies at most on the result of only two of these. If a fault affects binary SVM of which the result does not matter, the fault is tolerated, which may be the reason of the increased reliability of the multiclass SVM.

It is important to notice that these results are preliminary and, while they raise the discussion and present some insights, there are still some open questions. First, only two datasets were used for these SVMs. Different datasets possibly yield different levels of reliability. For instance, in datasets where the input vectors lie closer to the classification border, it is possible that faults are more likely to lead to failure. Minor perturbations on the weights of the classification function may more easily be noticeable in these cases. A second question is how different architectures impact on the reliability of SVMs. In this work, only combinatorial implementations were used. Alternative architectures, such as pipeline implementations, could affect how faults interact with systems. Furthermore, while both FPGA SVM implementations in this work presented intrinsic fault tolerance, this level may not be enough for critical systems. Thus, investigating further hardening techniques may be necessary in these cases.

8.2 Contributions on Low-Power Processor Implementations of Machine Learning

In Chapters 5 and 6, the reliability of low-power processor implementation of ML models were investigated. In Chapter 5, the assessment of a SVM and an ANN is conducted. The evaluation followed the same pattern used for the FPGA implementations, as both a fault emulation and a radiation campaign were conducted. The fault emulation campaign was performed by emulating failures in the registers of the processor used. In this situation, the SVM presented a slightly advantage over the ANN in terms of reliability, as it crashed less, while also masking, i.e. No failure, and tolerating more faults than its counterpart. Furthermore, the majority of the faults that were not masked led to Computing Crashes, which may be less harmful than a Critical Failure. A Computing Crash is easily detectable. While it is definitely not ideal, a fail safe protocol may be put into place in this scenario. A critical failure, on the other hand, is silent and may cause the system to make an erroneous decision, which could potentially be more harmful. In this work, only the ANN implementations was evaluated under radiation. The fault profile corroborated that one observed in the fault injections.

A more profound radiation campaign was conducted in the work presented in Chapter 6. An assessment of low-power processor implementations of SVM, ANN and RF was conducted. It was the first work in the literature in which a RF implementation was irradiated. While the three were capable to mitigate faults, the results have shown that, among the three algorithms, RF presented the highest level of reliability. In fact, during the campaign, no misclassifications, i.e. no critical failures, were detected on the RF, suggesting it possibly is the most reliable between the three tested. It is worth mentioning that no protection mechanisms were implemented, with the tolerance level being to intrinsic construction of the algorithms. In terms of the reason for the RF presenting the highest reliability, a possibility is that, by construction, it has mechanisms that may provide redundancy. It is made by a collection of independent BDTs, and the collection as whole may compensate for a fault on one.

There are still open questions in regards to reliability of ML algorithms implemented in low-power processors. Further experiments are needed to explore parameters that may affect their reliability. For instance, different datasets may change how these three algo-

rithms behave when a fault is present. Also, the structures of an ANN and of a RF models are defined the designer before training. For ANNs, it is up for the designer to the designer to choose the number of layers in the model as well as the number of neurons in each layer. For RFs, the number of BDTs is also chosen by the designer. In these cases, the different configurations may lead to different levels of reliability. Furthermore, fault injection/emulation campaigns are needed to better map the most sensitive points in these implementations to allow for the design of efficient hardening mechanisms.

8.3 Perspectives: Reliable Machine Learning

At the rate that ML applications are evolving, the tendency is that they will permeate more and more our lives as time goes by. Autonomous cars, which have been theorize fora for so long now, are closer to become a reality than ever. Space applications are using ML as well. Thus, studies like the ones in this thesis must continue in order to better understand how these techniques behave under the effects of radiation induced faults from a safety and cost perspective. In addition, the platforms which are being used to embed ML in such systems are evolving as well. In this thesis, implementation using *Commercial off-the-shelf* (COTS) were studied, which are great alternatives for space application such as nanosatellites. On the other hand, some companies, such as Tesla, have been investing in the design of their own proprietary platform, developing their own custom ASICs. Evidently, these novel platforms must be validated, but understanding the intricacies of the algorithms may greatly aid designers to develop more reliable platforms, which potentially could be achieved through testing COTS as well. For instance, in Chapter 6, it was shown that RF possibly, by construction, has and advantage over its counterparts. If these is further explored and confirmed, incorporating RFs in these systems could result in less need for redundancy, saving area and energy.

Presently, there is a lot of research (and hope) on the potential of ANNs. The ANN studied in this work is a classical example of the technique. Studying it allows for a comprehension of core strengths and issues of the techniques. However, modern implementations of ANNs, specially the ones used for image processing are much more complex in terms of structure. In fact, they may come in various shapes and sizes, with layers that are not necessarily composed by the classic neuron, feedback nodes and other variations. They are all built following the foundations of the classical implementations, thus making the assessing it possibly highlight issues or raise ideas that could benefit all variations of ANNs, even though they should each be studied as well.

Bibliography

[1] D. Binder, E. C. Smith, and A. B. Holman. Satellite anomalies from galactic cosmic rays. *IEEE Transactions on Nuclear Science*, 22(6):2675–2680, 1975.

[2] E. Normand. Single event upset at ground level. *IEEE Transactions on Nuclear Science*, 43(6):2742–2750, December 1996.

[3] Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artif. Intell.*, 134(1–2):57–83, January 2002.

[4] R. Behringer, S. Sundareswaran, B. Gregory, R. Elsley, B. Addison, W. Guthmiller, R. Daily, and D. Bevly. The darpa grand challenge - development of an autonomous vehicle. In *IEEE Intelligent Vehicles Symposium, 2004*, pages 226–231, 2004.

[5] Abigail C. Allwood, Joel A. Hurowitz, Benton C. Clark, Luca Cinquini, Scott Davidoff, Robert W. Denise, W. Timothy Elam, Marc C. Foote, David T. Flannery, James H. Gerhard, John P. Grotzinger, Christopher M. Heirwegh, Christina Hernandez, Robert P. Hodyss, Michael W. Jones, John Leif Jorgensen, Jesper Henneke, Peter R. Lawson, Yang Liu, Haley MacDonald, Scott M. McLennan, Kelsey R. Moore, Marion Nachon, Peter Nemere, Lauren O'Neil, David A. K. Pedersen, Kimberly P. Sinclair, Michael E. Sondheim, Eugenie Song, Nicholas R. Tallarida, Michael M. Tice, Alan Treiman, Kyle Uckert, Lawrence A. Wade, Jimmie D. Young, and Payam Zamani. The pixl instrument on the mars 2020 perseverance rover, 2021.

[6] Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Errors in Semiconductor Devices, January 2012.

[7] Rémi Gaillard. *Single Event Effects: Mechanisms and Classification*, pages 27–54. Springer US, Boston, MA, 2011.

[8] E. Normand and T. J. Baker. Altitude and latitude variations in avionics SEU and atmospheric neutron flux. *IEEE Transactions on Nuclear Science*, 40(6):1484–1490, December 1993.

[9] F. Villa, M. Baylac, S. Rey, O. Rossetto, W. Mansour, P. Ramos, R. Velazco, and G. Hubert. Accelerator-Based Neutron Irradiation of Integrated Circuits at GENEPI2 (France). In *IEEE Radiation Effects Data Workshop (REDW)*, pages 1–5, July 2014.

[10] C. Constantinescu. Neutron SER characterization of microprocessors. In *International Conference on Dependable Systems and Networks (DSN)*, pages 754–759, June 2005.

[11] Ronald Aylmer Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, September 1936.

[12] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[13] David Harrison and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5(1):81–102, 1978.

[14] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[15] James L McClelland, David E Rumelhart, PDP Research Group, et al. *Parallel distributed processing*, volume 2. MIT press Cambridge, MA, 1986.

[16] Jintaek Kang, Youngmin Yi, Kwanghyun Chung, and Soonhoi Ha. Nnsim: Fast performance estimation based on sampled simulation of gpgpu kernels for neural networks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, 2018.

[17] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[18] Ruizhou Ding, Zeye Liu, Ting-Wu Chin, Diana Marculescu, and R. D. Shawn Blanton. Flightnns: Lightweight quantized deep neural networks for fast and accurate inference. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[19] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[20] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer Science & Business Media, June 2013. Google-Books-ID: EqgACAAAQBAJ.

[21] I. Garali, M. Adel, S. Bourennane, and E. Guedj. Histogram-Based Features Selection and Volume of Interest Ranking for Brain PET Image Classification. *IEEE Journal of Translational Engineering in Health and Medicine*, 6, Art. No. 2100212, 2018.

[22] C. Weinrich, C. Vollmer, and H. M. Gross. Estimation of human upper body orientation for mobile robotics using an SVM decision tree on monocular images. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2147–2152, October 2012.

[23] I. Saha, U. Maulik, S. Bandyopadhyay, and D. Plewczynski. SVMeFC: SVM Ensemble Fuzzy Clustering for Satellite Image Segmentation. *IEEE Geoscience and Remote Sensing Letters*, 9(1):52–55, January 2012.

[24] M. Ruiz-Llata, G. Guarnizo, and M. Yébenes-Calvino. FPGA implementation of a support vector machine for classification and regression. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–5, July 2010.

[25] P. Hazucha, T. Karnik, J. Maiz, S. Walstra, B. Bloechel, J. Tschanz, G. Dermer, S. Hareland, P. Armstrong, and S. Borkar. Neutron soft error rate measurements in a 90-nm CMOS process and scaling trends in SRAM from 0.25-/spl mu/m to 90-nm generation. In *IEEE International Electron Devices Meeting*, pages 21.5.1–21.5.4, December 2003.

[26] F. Libano, P. Rech, L. Tambara, J. Tonfat, and F. Kastensmidt. 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.

[27] 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*, 66(1):216–222, January 2019.

[28] 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*, pages 1–15, 2018.

[29] A. Coelho, R. Laurent, M. Solinas, J. Fraire, E. Mazer, N. E. Zergainoh, S. Karaoui, and R. Velazco. On the Robustness of Stochastic Bayesian Machines. *IEEE Transactions on Nuclear Science*, 64(8):2276–2283, August 2017.

[30] J. C. Wang, L. X. Lian, Y. Y. Lin, and J. H. Zhao. VLSI Design for SVM-Based Speaker Verification System. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(7):1355–1359, July 2015.

[31] F. Ye, F. Firouzi, Y. Yang, K. Chakrabarty, and M. B. Tahoori. On-Chip Droop-Induced Circuit Delay Prediction Based on Support-Vector Machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(4):665–678, April 2016.

[32] M. Papadonikolakis and C. S. Bouganis. A Heterogeneous FPGA Architecture for Support Vector Machine Training. In *18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 211–214, May 2010.

[33] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

[34] L. Acunha Guimarães, T. Ferreira de Paiva Leite, R. Possamai Bastos, and L. Fesquet. Non-Intrusive Testing Technique for Detection of Trojans in Asynchronous Circuits. In *DATE*, 2018.

[35] Zynq-7000 All Programmable SoC.

[36] M. Solinas, A. Coelho, J. A. Fraire, N. E. Zergainoh, P. A. Ferreyra, and R. Velazco. Preliminary results of netfi-2: An automatic method for fault injection on HDL-based designs. In *18th IEEE Latin American Test Symposium (LATS)*, pages 1–4, March 2017.

[37] W. Mansour and R. Velazco. An automated seu fault-injection method and tool for HDL-based designs. *IEEE Transactions on Nuclear Science*, 60(4):2728–2733, Aug 2013.

[38] W. Mansour, R. Velazco, R. Ayoubi, H. Ziade, and W. El Falou. A method and an automated tool to perform set fault-injection on HDL-based designs. In *25th International Conference on Microelectronics (ICM)*, pages 1–4, Dec 2013.

[39] L. A. Tambara, P. Rech, E. Chielle, J. Tonfat, and F. L. Kastensmidt. Analyzing the Impact of Radiation-Induced Failures in Programmable SoCs. *IEEE Transactions on Nuclear Science*, 63(4):2217–2224, August 2016.

[40] D. Rossi, M. Omana, F. Toma, and C. Metra. Multiple transient faults in logic: an issue for next generation ICs? In *20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 352–360, Oct 2005.

[41] E. Nurvitadhi, and D. Sheffield, A. Mishra, S. Krishnan, and D. Marr. Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. In *26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, August 2016.

[42] M. G. Trindade, A. Coelho, C. Valadares, R. A. C. 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*, 66(7):1441–1448, July 2019.

[43] C. Weulersse, S. Houssany, N. Guibbaud, J. Segura-Ruiz, J. Beaucour, F. Miller, and M. Mazurek. Contribution of thermal neutrons to soft error rate. *IEEE Transactions on Nuclear Science*, 65(8):1851–1857, Aug 2018.

[44] Fabio Benevenuti and Fernanda Lima Kastensmidt. Comparing exhaustive and random fault injection methods for configuration memory on SRAM-based FPGAs. In *2019 IEEE Latin American Test Symposium (LATS)*, pages 87–92, March 2019.

[45] Robert Le. Soft error mitigation using prioritized essential bits, April 2012. Application Note, XAPP538.

[46] M. G. Trindade, A. Coelho, C. Valadares, R. A. C. 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*, 66(7):1441–1448, July 2019.

[47] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen 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*, pages 1–12, November 2017.

[48] M. A. Neggaz, I. Alouani, S. Niar, and F. Kurdahi. Are CNNs reliable enough for critical applications? an exploratory study. *IEEE Design & Test*, 37(2):76–83, 2020.

[49] F. F. d. 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*, 68(2):663–677, June 2019.

[50] Felipe Rocha da Rosa, Rafael Garibotti, Luciano Ost, and Ricardo Reis. 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.

[51] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. Ares: A framework for quantifying the resilience of deep neural networks. In *Design Automation Conference (DAC)*, pages 1–6, June 2018.

[52] R. Possamai Bastos, Y. Monnet, G. Sicard, F. Kastensmidt, M. Renaudin, and R. Reis. Comparing transient-fault effects on synchronous and on asynchronous

circuits. In *IEEE International On-Line Testing Symposium (IOLTS)*, pages 29–34, June 2009.

[53] Mohammad Shokrolah-Shirazi and Seyed Ghassem Miremadi. Fpga-based fault injection into synthesizable verilog hdl models. In *2008 Second International Conference on Secure System Integration and Reliability Improvement*, pages 143–149, 2008.

[54] M. Solinas, A. Coelho, J. A. Fraire, N. E. Zergainoh, P. A. Ferreyra, and R. Velazco. Preliminary results of netfi-2: An automatic method for fault injection on hdl-based designs. In *2017 18th IEEE Latin American Test Symposium (LATS)*, pages 1–4, 2017.

[55] Alexander Aponte-Moreno, Felipe Restrepo-Calle, and Cesar Pedraza. Mifit: A fault injection tool to validate the reliability of microprocessors. In *2019 IEEE Latin American Test Symposium (LATS)*, pages 1–5, 2019.

[56] M. De Carvalho, D. Sabena, M. Sonza Reorda, L. Sterpone, P. Rech, and L. Carro. Fault injection in gpgpu cores to validate and debug robust parallel applications. In *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, pages 210–211, 2014.

List of Figures

List of Tables

9

List of Publications and Presentations

9.1 International Journals

1. **TRINDADE, M. G.**; BENEVENUTI, F. LETICHE, M.; BEAUCOUR, J.; KASTENSMIDT, F.; POSSAMAI BASTOS, R.; Effects of thermal neutron radiation on a hardware-implemented machine learning algorithm. Elsevier Microelectronics Reliability Journal, January 2021.

2. POSSAMAI BASTOS, R.; DUTERTRE, J.M.; **TRINDADE, M. G.**; VIERA, R. A. C.; POTIN, O.; LETICHE, M.; CHEYMOL, B.; BEAUCOUR, J.; Assessment of On-Chip Current Sensor for Detection of Thermal-Neutron Induced Transients. IEEE Transactions on Nuclear Science, TNS, July 2020.

3. GUAZZELLI, R. A.; **TRINDADE, M. G.**; GUIMARÃES, L. A.; LEITE, T. F. P.; FESQUET, L.; POSSAMAI BASTOS, R.; Trojan Detection Test for Clockless Circuits. Springer Journal of Electronic Testing: Theory and Applications, JETTA, February 2020.

4. **TRINDADE, M. G.**; COELHO, A.; VALADARES, C.; VIERA, R. A. C.; REY, S.; CHEYMOL, B.; BAYLAC, M.; VELAZCO, R.; POSSAMAI BASTOS, R.; Assessment of a Hardware-Implemented Machine Learning Technique under Neutron Irradiation. IEEE Transactions on Nuclear Science, TNS, June 2019.

5. GUAZZELLI, R. A.; **TRINDADE, M. G.**; FESQUET, L.; POSSAMAI BASTOS, R. Learning-Based Reliability Assessment Method for Detection of Permanent Faults in Clockless Circuits. Elsevier Microelectronics Reliability Journal, September 2019.

9.2 Conferences

1. BANDEIRA, V.; SAMPFORD, J.; GARIBOTTI, R.; **TRINDADE, M. G.;** POSSAMAI BASTOS, R.; REIS, R.; OST, L.; Impact of Radiation-Induced Soft Error on Embedded Cryptography Algorithms. 32nd European Symposium on Reliability of Electron Devices, Failure Physics and Analysis, October 2021.

2. **TRINDADE, M. G.**; GARIBOTTI, R.; OST, L.; LETICHE, M.; BEAUCOUR, J.;

POSSAMAI BASTOS, R.; Assessment of Machine Learning Algorithms for Near-Sensor Computing under Radiation Soft Errors. 27th IEEE International Conference on Electronics Circuits and Systems, November 2020.

3. POSSAMAI BASTOS, R.; DUTERTRE, J.M.; **TRINDADE, M. G.**; VIERA, R. A. C.; POTIN, O.; LETICHE, M.; CHEYMOL, B.; BEAUCOUR, J.; Assessment of Current Sensor On-Chip for Detecting Neutron-Induced Transients via Body Terminals. 30th European Conference on Radiation and its Effects on Components and Systems (RADECS), September 2019.

4. GUAZZELLI, R. A.; **TRINDADE, M. G.**; FESQUET, L.; POSSAMAI BASTOS, R. Learning-Based Reliability Assessment Method for Detection of Permanent Faults in Clockless Circuits. 30th European Symposium on Reliability of Electron Devices, September 2019.

5. **TRINDADE, M. G.**; POSSAMAI BASTOS, R. On the reliability of Support Vector Machiness. Journées Nationales du Réseau Doctoral en Micro-nanoélectronique, June 2019.

6. **TRINDADE, M. G.**; COELHO, A.; VALADARES, C.; VIERA, R. A. C.; REY, S.; CHEYMOL, B.; BAYLAC, M.; VELAZCO, R.; POSSAMAI BASTOS, R.; Assessment of Hardware-Implemented Support Vector Machine under Radiation Effects. 29th European Conference on Radiation Effects on Components and Systems (RADECS), September 2018.

10

Acronym List

AGC *Apollo Guidance Computer* 2

AI *Artificial Intelligence* 1–4, 75

ANN *Artificial Neural Network* 4, 5, 19, 28, 41, 44, 48, 60, 61, 65, 68–72, 75, 76, 81, 96–98, 109

ASIC *Application-Specific Integrated Circuit* 29, 85, 98

AXI *Advanced eXtensible Interface* 34, 50, 51, 56, 58

BDT *Binary Decision Trees* 23, 24, 76, 80, 81, 96, 97, 108

BRAM *Block RAM memory* 28, 50, 52, 53, 55

CLB *Configuration Logic Block* 28, 52, 53

CNN *Convolutional Neural Network* 44, 60, 61, 65

CORDIC *COordinate Rotation DIgital Computer* 29, 30

COTS *Commercial off-the-shelf* 98

CRAM *Configuration Random-Access Memory* 51–53

DARPA *Defense Advanced Research Projects Agency* 3

DNN *Deep Neural Network* 65

DRAM *Dynamic Random Access Memory* 12

DUT *Design Under Test* 49–51, 53–55

EDIF *Electronic Design Interchange Format* 34

FF *Flip-Flop* 28, 50

FIT *Failure in Time* 12, 13, 40, 58

FPGA *Field-Programmable Gate Array* 3–5, 12, 19, 28, 29, 31–34, 36–41, 43, 44, 49–54, 57–61, 65, 84, 85, 94–96, 108, 109

GDB *GNU Debugger* 66, 67, 69, 70, 84–87, 91

GPU *Graphic Processor Unit* 3, 19, 41, 44, 60, 61, 84, 85

IC *Integrated Circuit* 2, 29–31

ICAP *Internal Configuration Access Port* 51, 52

ILL *Institut Laue-Languevin* 56, 71

IP *Intellectual Property* 50, 51, 57

LPSC *Laboratory of Subatomic Physics & Cosmology* 38

LUT *Look-Up Table* 28, 32, 34–37, 50

MAC *Multiply-ACcumulate* 20, 80

ML *Machine Learning* vii, 3–5, 16, 41, 65–72, 74, 75, 77–81, 84, 86, 87, 91, 94, 96, 98, 109, 112

MODNET *MODify NETlist* 33–35

NQKF *Novel Quartenion Kalman Filter* 84, 87, 88, 90, 91

OS *Operating System* 89

PAC-G *Platform for Advanced Characterisation* 56

PIXL *Planetary Instrument for X-ray Lithochemistry* 3

PL *Programmable Logic* 50

PS *Processing System* 50, 56, 58

RF *Random Forest* 4, 5, 23, 24, 75, 76, 78–81, 96–98

RTL *Register-Transfer Level* 85

SEFI *Single Event Functional Interrupt* 10

SEL *Single Event Latch-Up* 10

SEM *Soft Error Mitigation* 57

SER *Soft Error Rate* 13

SEU *Single Event Upset* 28, 50, 51, 53–55, 78

SMO *Sequential Minimal Optimization* 21–23

SoC *System-on-Chip* 48, 56

SRAM *Static Random Access Memory* 28, 38, 50

SUT *System Under Test* 66, 67, 69–71, 75–79, 109

SV *Support Vector* 29, 30

SVM *Support Vector Machine* vi, vii, 4, 5, 21–23, 26, 28–41, 43, 44, 46–51, 53–61, 65, 68–72, 75, 76, 78–81, 94–96, 108, 109, 112

UAVs *Unmanned Aerial Vehicles* 88

USA *United States of America* 2

USSR *Union of Soviet Socialist Republics* 2

Assessement of Edge Machine-Learning Systems under Radiation-Induced Effect

Résumé

Les algorithmes d'apprentissage automatique (ML) ont gagné popularité ces dernières années, en fournissant des solutions simples à une large gamme d'applications, comme les moteurs de recherche, les systèmes de recommandation, la robotique et même les voitures autonomes. Comme le dernier exemple suggère, le ML a gagné de la place même sur les systèmes critiques, y compris les applications au niveau terrestre, avioniques et spatiales. Toutefois, certaines exigences doivent être respectées dans de tels systèmes, vu que les fautes sont soit coûteuses, soit désastreuses (ou les deux) dans ces scénarios. Premièrement, ces systèmes ont souvent une puissance de computation limitée sur leurs composants. Deuxièmement, les fautes induit par la radiation sont également une préoccupation majeure, spécialement aux altitudes avioniques et dans l'espace, mais toujours suffisamment pertinents pour les applications au niveau terrestre. En explorant ces requis, cette thèse évalue les effets de la radiation sur des implémentations d'algorithmes d'apprentissage automatique en périphérie proéminentes. Initialement, des implémentations FPGA de l'algorithme Support Vector Machine (SVM) ont été évaluées sous les effets de la radiation des neutrons rapides et thermiques, des particules qui doivent être pris en compte surtout pour les applications au sol et avioniques. Cette évaluation a été complémenté par deux différentes techniques d'injection des fautes pour mieux comprendre les effets des fautes sur les systèmes. Ces tests ont montré que les implémentations aient un certain niveau de tolérance intrinsèque aux fautes. Suite à ce travail, les implémentations de trois algorithmes - Réseaux de neurones artificiels (ANN), Random Forest (RF) et SVM – implémentés dans des microcontrôleurs commerciaux (cartes de développement STM32 Nucleo) ont été évaluées sous les effets des neutrons rapides. Encore une fois, en suivant la tendance observée sur les implémentations FPGA, ils ont présenté une tolérance intrinsèque aux fautes. En plus, l'implémentation du RF a pu tolérer toutes les fautes induites par la radiation. Ce travail a également été complémenté par une campagne d'injection de fautes. L'outil utilisé pour ce dernier a été développé dans le cadre de la thèse.

Mots-clés : Apprentissage Automatique; Informatique en Périphérie ; Effets de la Radiation; Systèmes Embarqués

Abstract

Machine learning (ML) algorithms have grown in popularity in recent years, providing straightforward solutions to a wide range of applications, such as search engines, recommendation systems, robotics and even self-driven cars. As the last example suggests, ML has been gaining its space even on critical systems, including ground, avionics and space applications. However, there are requirements that need to be met in such systems, as faults are either costly or disastrous (or both) in these scenarios. First, often these systems have limited processing power on their embedded components. Secondly, radiation induced faults are also a major concern, specially at avionics altitudes and on space, but still relevant enough for ground level applications. Exploring these requirements, this thesis evaluates the radiation effects on edge implementation of prominent machine learning algorithms. Initially, FPGA implementations of Support Vector Machine (SVM) algorithm have been evaluated under the effects of both fast and thermal neutron radiation, particles that should be considered for ground and avionic applications. This evaluation was complemented using two different fault injection techniques to better understand the effects of faults on the systems. These tests have shown that the implementations had a certain level of intrinsic fault tolerance. Following this work, the implementations of three Algorithms - Artificial Neural Networks (ANN), Random Forest (RF) and SVM - using off-the-shelf micro-controllers (STM32 Nucleo development boards) have been evaluated under the effects of fast neutrons. Again, following the trend observed on the FPGA implementations, they presented intrinsic fault tolerance. Furthermore, the RF implementation was able to tolerate all the radiation induced failures. This work was also complemented by a fault injection campaign. The tool used was developed within the context of the thesis.

Keywords : Machine Learning; Edge Computing; Radiation Effects; Embedded Systems