POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Understanding Non-volatile Memories in Space : A Deployment Experience

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Eleonora Giudici**

Student ID: 221638
Advisor: Prof. Luca Mottola
Academic Year: 2024-2025

Abstract

Nanosatellites are increasingly used in scientific, commercial, and educational missions due to their low cost and rapid development cycle. They often contain Commercial Off-The-Shelf (COTS) components, vulnerable to cosmic rays and space radiation, which can cause transient or permanent hardware faults and compromise mission performance.

This thesis analyzes the reliability of non-volatile memories in nanosatellites, focusing on FRAM, ReRAM, and MRAM. These memories are critical because nanosatellites operate intermittently, being powered by solar panels. They shut down when power is unavailable and restart when power is restored, making it necessary to preserve data between activation cycles.

We studied memory faults onboard a nanosatellite in Low Earth Orbit (LEO), considering the frequency, spatial distribution, and evolution of bit-flips. Faults typically affect consecutive memory cells and can be transient or permanent. FRAM exhibits multiple phases with different levels of corruption, as well as permanent and transient bit-flips. ReRAM exhibits the highest total number of bit-flips, with fewer permanent faults than FRAM. Finally, MRAM proves to be the most robust, with most faults being short-lived and only one permanent bit-flip.

Based on these observations, we developed fault models based on statistical methods and a machine learning approach using a Variational AutoEncoder (VAE), both evaluated on their ability to generate faults with characteristics similar to those observed in orbit. The results show that the statistical models accurately replicate the distribution and persistence of faults, while the autoencoder captures spatial correlations and can introduce faults in previously unaffected memory regions, although without fully reproducing all evaluation metrics. These models enable realistic fault injection, supporting the design and reliability evaluation of non-volatile memory systems aboard nanosatellites.

Keywords: Nanosatellite, Non-volatile memory, FRAM, ReRAM, MRAM, Memory faults, Fault models

Abstract in lingua italiana

I nanosatelliti sono sempre più utilizzati in missioni scientifiche, commerciali e didattiche grazie al basso costo e al rapido ciclo di sviluppo. Spesso contengono componenti off-the-shelf (COTS), vulnerabili ai raggi cosmici e alle radiazioni spaziali, che possono causare guasti hardware transitori o permanenti e compromettere le prestazioni della missione.

Questa tesi analizza l'affidabilità delle memorie non volatili nei nanosatelliti, concentrandosi su FRAM, ReRAM e MRAM. Queste memorie sono fondamentali perché i nanosatelliti funzionano a intermittenza, essendo alimentati da pannelli solari. Si spengono in assenza di alimentazione e si riaccendono al ripristino dell'alimentazione, rendendo necessaria la conservazione dei dati tra i cicli di attivazione.

Abbiamo studiato i guasti della memoria a bordo di un nanosatellite in orbita terrestre bassa (LEO), considerando la frequenza, la distribuzione spaziale e l'evoluzione dei bit-flip. I guasti generalmente interessano celle di memoria consecutive e possono essere transitori o permanenti. La FRAM presenta più fasi con diversi livelli di corruzione, nonché bit-flip permanenti e transitori. La ReRAM presenta il numero totale più elevato di bit-flip, con meno guasti permanenti rispetto alla FRAM. Infine, la MRAM si dimostra la più robusta, con la maggior parte dei guasti di breve durata e un solo bit-flip permanente.

Sulla base di queste osservazioni, abbiamo sviluppato modelli di guasto basati su metodi statistici e un approccio di apprendimento automatico con un autoencoder variazionale (VAE), entrambi valutati in base alla capacità di generare guasti con caratteristiche simili a quelli osservati in orbita. I risultati mostrano che i modelli statistici replicano accuratamente distribuzione e persistenza dei guasti, mentre l'autoencoder cattura le correlazioni spaziali e può introdurre guasti in regioni di memoria precedentemente non interessate, sebbene senza riprodurre completamente tutte le metriche di valutazione. Questi modelli consentono un'iniezione di guasti realistica, supportando la progettazione e la valutazione dell'affidabilità dei sistemi con memorie non volatili a bordo di nanosatelliti.

Parole chiave: Nanosatellite, Memoria non volatile, FRAM, ReRAM, MRAM, Guasti di memoria, Modelli di guasto

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

1 Introduction **1**
1.1 Problem and Contribution 2
1.2 Thesis Structure 5

2 Background **7**
2.1 Nanosatellites 7
2.2 Radiation-induced Faults 10
2.2.1 Permanent Faults 10
2.2.2 Transient Faults 11
2.3 Non-volatile Memories in Space 11
2.3.1 Flash Memory 12
2.3.2 FRAM 14
2.3.3 MRAM 15
2.3.4 ReRAM 17
2.3.5 PCM 18
2.4 Fault Injection Techniques 19
2.4.1 Hardware Fault Injection 20
2.4.2 Software Fault Injection 21
2.4.3 Simulation-Based Fault Injection 21
2.4.4 Hybrid Fault Injection 22
2.5 Generative Neural Networks 23
2.5.1 GAN 23
2.5.2 VAE 25

3 Problem Statement 27
3.1 Data Collection . . . 27
3.2 Data Analysis . . . 28
3.3 Fault Models . . . 29

4 Experimental Setup 31
4.1 Nanosatellite Architecture . . . 31
4.2 Mission . . . 34

5 Analysis Framework 35
5.1 Data Preparation Workflow . . . 35
5.1.1 Data Cleaning . . . 36
5.1.2 Descriptive Analysis . . . 37
5.1.3 Visualization . . . 38
5.1.4 Features Extraction . . . 39
5.2 Fault Classification Procedures . . . 40
5.3 Dependency Between Faults . . . 41
5.4 Threats to Validity . . . 42

6 Results and Discussion 45
6.1 Data Cleaning Results . . . 46
6.2 Descriptive Analysis Results . . . 48
6.2.1 Evolution of Bit Flips . . . 48
6.2.2 Mean and Standard Deviation . . . 49
6.3 Features Extraction and Classification . . . 50
6.3.1 Fault Distribution . . . 51
6.3.2 Block Dependency Analysis . . . 52
6.3.3 Permanent Faults Analysis . . . 53
6.3.4 Transient Faults Analysis . . . 55
6.3.5 Heatmaps . . . 60
6.4 Memory Comparison . . . 61

7 Fault Injection 65
7.1 Statistical Approach . . . 65
7.1.1 Static Fault Generation . . . 66
7.1.2 Sequential Fault Generation . . . 66
7.2 Machine Learning . . . 67
7.2.1 Neural Network Design . . . 68

7.2.2 Data Generation 70
7.3 Interface Design 71

8 Evaluation **73**
8.1 Metrics Definition 73
8.2 Statistical Comparison 74
8.3 Parameters Selection 76
8.4 Results 78
8.4.1 Statistical Models Results 78
8.4.2 Machine Learning Results 79

9 Suggestions for Future Missions **81**
9.1 Absolute Timestamps 81
9.2 Dataset Size 82
9.3 Onboard Algorithm Execution 82

10 Conclusion and Future Works **85**

Bibliography **89**

List of Figures **95**

List of Tables **97**

Acknowledgements **99**

1 | Introduction

In the last decade, the race to space has been intensifying [49], leading to the development of satellites equipped with advanced technologies onboard by both governmental agencies and private companies.

These large-scale satellites represent significant technological achievements, but they come with some issues. The first challenge is the high cost. Launching these massive structures requires specialized launch vectors and complex logistics. The manufacturing process demands extensive manpower and advanced expertise, further increasing the overall investment needed. Moreover, the development and assembly of these sophisticated systems may take many years, often spanning over a decade.

Because of these limitations, there has been a growing interest in developing more cost-effective and accessible alternatives and nanosatellites, particularly CubeSats, have become an increasingly popular solution [58]. Their small size, low cost, and rapid development cycles make them an attractive option for academic research, Earth observation, technology demonstrations, and commercial applications [68]. These smaller satellites have the advantage of being lightweight and compact, enabling them to be launched in groups or as secondary payloads, thus significantly reducing launch costs. By lowering financial and logistical barriers, more institutions, including universities and startups, can participate in space exploration and research. This trend is reshaping the satellite industry, enabling the rapid deployment of new technologies.

Despite their advantages, nanosatellites face several challenges. The main issue is that environmental factors such as space radiation, cosmic rays, and extreme temperature fluctuations can significantly impact their performance and reliability [9]. To maintain a small size, low power consumption, and reduced cost, nanosatellites typically use commercial-off-the-shelf (COTS) components, which are not always designed to operate under extreme space conditions [25, 64]. High-energy particles from the Sun or deep space can interact with these components, causing transient or permanent damage. Temperatures in space can swing drastically between sunlight and shadow, placing additional stress on materials and systems. Furthermore, their limited surface area and mass prevent the inclusion of

heavy shielding or complex thermal control systems.

These vulnerabilities increase the likelihood of faults occurring in electronic components, especially memory units, compromising stored data or, worse, the functioning of the entire system. Understanding and detecting these faults is essential to ensure the reliability of nanosatellite missions. If not properly managed, these faults can lead to mission-critical failures, data loss, or system crashes.

1.1. Problem and Contribution

One of the main challenges with nanosatellites is their vulnerability to the space environment, particularly due to cosmic rays and radiation that can strike the hardware architecture, damage electronic components, and cause faults [9]. Many studies have been conducted in this field, but existing approaches are typically based on ground experiments, which can only approximate real faults and do not faithfully replicate those observed in orbit [59].

Our work focuses on studying memory faults in non-volatile memories recorded during the actual space mission of a nanosatellite. We chose to focus on non-volatile memories because nanosatellites are powered by solar panels and exhibit intermittent behavior, repeatedly shutting down and restarting depending on solar energy availability. When solar energy is unavailable, the nanosatellite temporarily shuts down, so it is essential to have onboard memories capable of storing data even when the system is not powered.

The three main challenges follow. This thesis focuses on the last two:

- **Data collection:** designing an experiment to acquire data on memory faults, including the setup of instrumentation and conditions necessary to capture fault occurrences in non-volatile memories during the mission.
- **Data analysis**: analyzing the collected data to identify fault patterns observed in space and to understand their distributions and dynamics.
- **Fault models creation:** developing fault models [50] that realistically reproduce the observed fault patterns, accurately reflecting the dynamics identified through the analysis.

A real nanosatellite was built following the standardized Cubesat form [5], and its structure is shown in Figure 1.1. Among other components for parallel experiments, it includes a dedicated board equipped with three selected non-volatile memories: a Ferroelectric Random-Access Memory (FRAM), a Magnetoresistive Random-Access Memory

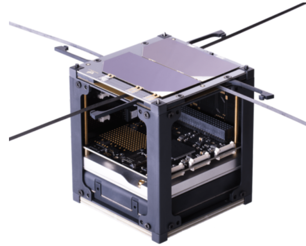


Figure 1.1: CubeSat external structure.

(MRAM), and a Resistive Random-Access Memory (ReRAM). Each memory uses a different technology to retain its logical state in the absence of power, making them suitable for low-power systems due to their minimal energy consumption.

The nanosatellite was launched on November 4th, 2024. It was deployed at 732 km from Earth with an initial attitude and velocity expected to ensure roughly three months of operation. These initial estimates were far exceeded, as the satellite remained operational until the end of February 2025. The nanosatellite is powered by solar panels [16], and once sufficient energy is accumulated, it executes a well-defined program. After each execution, it transmits to Earth a memory dump, which is a binary sequence representing the logical state of each cell in the respective non-volatile memory at that specific moment. Each memory dump is then compared with a reference sequence, which represents the correct values that should be stored in memory, and bit flips are detected. For each memory, we organize the memory dumps in the order they were received from the nanosatellite and analyze the evolution of bit flips over time. The mean and standard deviation of bit flips across dumps are then computed to assess the overall reliability of each memory under space radiation.

We observed that FRAM shows distinct phases in the evolution of bit flips, each with different means and standard deviations, which gradually decrease before stabilizing in a manner similar to MRAM. ReRAM exhibits the highest absolute number of bit flips; however, due to its large storage capacity, the percentage of corrupted data remains very low. MRAM proves to be the most robust technology, displaying stable behavior and offering the best compromise between the mean number of bit flips and storage capacity.

Analyzing the distribution of bit flips, we identified recurring fault patterns: flips typically affect consecutive memory cells, ranging from a single bit to multiple adjacent bits. This behavior is similar to that observed in literature for ground radiation experiments on non-volatile memories employing the same technologies we used in our study [43].

These faults can be classified into two categories: transient faults, which disappear after

some time, and permanent faults, which persist and indicate hardware damage that forces a cell into a fixed logical state [76].

We observed that permanent faults are rare and typically consist of isolated bits stuck at logical '1'. FRAM exhibited 5 permanent bit flips, ReRAM 3, and MRAM only 1, making MRAM the most robust to this type of fault. These results differ from those reported in literature [43], where no permanent faults were observed after exposing the memories to heavy ions in laboratory experiments. This demonstrates that results from ground experiments do not always match what can be observed under actual in-orbit conditions.

In contrast, for transient faults, no logical state appears more vulnerable than the other, with flips occurring with similar probability on both '0' and '1'. In all memories, short sequences of bit flips are more common than long ones, and in some cases faults persist across multiple memory dumps before disappearing. MRAM faults tend to be short-lived, often vanishing after a single dump, whereas FRAM and ReRAM exhibit faults that last longer.

Across all three memories, aside from a few isolated permanent bit flips, no specific regions proved more vulnerable than others. Overall, the average corruption rate stayed below 0.1%, confirming that all three memory technologies demonstrate high robustness in space environments.

Based on this study, we developed fault models that capture the main statistical characteristics of the observed faults. These models reflect the actual behavior of radiation effects on the electronic devices of the three selected non-volatile memories observed in space. We adopted two different approaches:

- **Statistical methods:** aim to reproduce observed faults according to their statistical properties. Each fault is inserted based on the frequency with which it was observed, ensuring that the model accurately reflects their distributions while preserving their size and persistence consistent with the experimental data. Two variants are considered: a *static model*, which introduces faults without accounting for their persistence across consecutive memory dumps, and a *sequential model*, which also considers the persistence of faults, reproducing correlations observed in consecutive memory dumps.

- **Machine learning:** by training a *Variational Autoencoder (VAE)* with the collected data, it is possible to generate synthetic faults. This approach allows the creation of new fault patterns that maintain the same statistical properties as those observed onboard the nanosatellite in orbit.

We apply each technique 200 times consecutively to generate different combinations of faults, simulating those observed in the memory dumps. From these results, we compute the mean and standard deviation of the number of generated bit flips to verify whether the models preserve the overall corruption levels. Using the *Wasserstein distance* [56], a metric that quantifies the difference between two probability distributions, we assess whether other statistical properties, such as the overall distributions of fault size and persistence, remain consistent with those observed in the original data. The distance is normalized to make the results comparable.

Both statistical models preserve the mean and standard deviation across all memories and maintain the distribution of fault sizes, but only the sequential model accurately reproduces fault persistence. The machine learning model can generate faults spatially distributed similarly to those observed and maintains a mean number of bit flips comparable to the real data. However, the standard deviation and other distributions related to fault size and persistence differ from those observed in the actual memory dumps. This indicates that the network has successfully learned spatial patterns but remains inaccurate, likely due to the limited size of the training dataset.

1.2. Thesis Structure

The following outline presents the organization of this thesis:

- **Chapter 2** describes necessary aspects to understand the thesis work, starting with an overview of nanosatellites, non-volatile memory technologies, faults that can occur due to exposure to radiation in orbit, and common fault injection techniques. An overview of the most common generative network models is provided.
- **Chapter 3** presents the problem addressed in this thesis, focusing on collecting and analyzing data and developing fault models that reproduce the faults observed in real scenarios.
- **Chapter 4** focuses on the experimental setup, describing the hardware architecture, the choice of non-volatile memories, the communication method with the ground, and other key aspects of the mission. Although this is not the main focus of the thesis, it is included to provide a complete overview of the experiment.
- **Chapter 5** describes the steps used to analyze the data, starting with the preparation phase, which is essential for the subsequent fault classification and reproduction.
- **Chapter 6** presents the results obtained using the analysis framework described

in the previous chapter and discusses them, providing a comparison between the three non-volatile memories onboard. These results allow a comparison among the memories under study to understand the advantages and limitations of each one.

- **Chapter 7** presents the methods used to reproduce faults based on real observations and statistics derived from the actual data received. These methods follow both a statistical approach and a machine learning approach.
- **Chapter 8** describes the methods and metrics used for evaluating the fault models and presents the results for each method described in the previous Chapter.
- **Chapter 9** contains a list of suggestions for future missions. This list is based on the observations made during our experiment and offers advice on how to improve the experiment to achieve more accurate analyses and fault injection techniques.
- **Chapter 10** provides an overview of what we achieved and learned from our experiment, covering both the data analysis and fault injection, along with ideas for future work.

2 | Background

In this chapter, we introduce the main aspects necessary to explain the contributions of this thesis.

We begin with an overview of nanosatellites in Section 2.1, focusing on their main characteristics, operational constraints, and challenges imposed by the space environment.

Then, Section 2.2 introduces a classification of radiation-induced faults that can affect devices in space, followed by an overview of the most common non-volatile memory technologies typically employed in low-power systems operating in such harsh environments in Section 2.3.

In Section 2.4, we present fault injection techniques typically used to analyze system responses under radiation stress.

Finally, Section 2.5 introduces generative machine learning frameworks that can be used to replicate the statistical properties of observed faults in non-volatile memories.

2.1. Nanosatellites

In recent years, the number of space missions has largely increased [49], including the launch of satellites by both governmental agencies and private companies.

Satellites are typically large in size and equipped with advanced technologies. Notable examples include the James Webb Telescope [19], designed for detailed observation of the universe, and the Hubble Space Telescope [12], which has provided invaluable insights into space for over three decades.

These satellites represent significant technological achievements, but there are also some issues, such as high costs and long development times. The high cost is because launching these massive structures requires specialized rockets and complex logistics, and the development and assembly of such complex systems can take a long time, sometimes even years. Additionally, the manufacturing process demands extensive manpower and advanced expertise, further increasing the overall investment required.

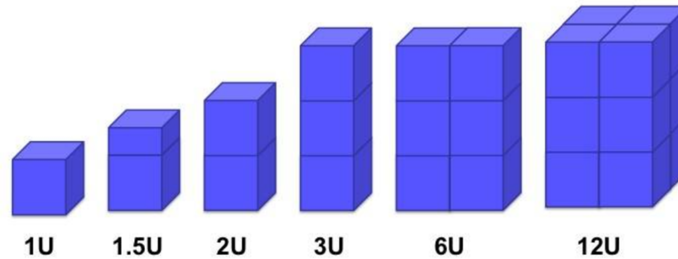


Figure 2.1: CubeSats common formats [5].

For these reasons, it became necessary to lower financial and logistical barriers and develop a solution to reduce costs and development time, allowing universities, startups, and companies to participate in space exploration and research. Consequently, there has been a growing interest in developing more cost-effective and accessible alternatives. For this reason, nanosatellites, particularly CubeSats, have become an increasingly popular solution [58].

Nanosatellites are a class of compact spacecraft known for their affordability, modular design, and rapid development timelines. While originally developed for educational purposes, they have evolved to support a wide range of applications. These include data relay, communication services, Earth observation, and in-orbit technology demonstrations [68].

An interesting and widely used type of nanosatellite is the CubeSat, which adheres to a standardized form factor of 1U (10 cm × 10 cm × 10 cm). This compact design enables faster development cycles compared to larger satellites, simplifying both payload integration and compatibility with a variety of launch systems [7].

Multiple 1U units can be combined, as illustrated in Figure 2.1, to create larger and more capable satellites. These multi-unit configurations provide additional space for instruments, greater power capacity, and increased operational autonomy, while maintaining relatively low costs thanks to their modular design.

To lower costs and reduce development time, COTS components are widely used. These are electronic devices and systems already available on the market, originally designed for terrestrial or industrial applications. Even though they were not specifically developed for space, COTS hardware is extensively used in nanosatellites due to its low cost, wide availability, and high performance [64]. In addition, COTS hardware provides nanosatellites

with more computing power compared to radiation-tolerant hardware [55].

Nanosatellites are typically launched into orbit using a variety of vehicles, including dedicated small launchers or rideshare opportunities on larger rockets.

The main orbital regions in which they operate are Low Earth Orbit (LEO), Medium Earth Orbit (MEO), and Geostationary Earth Orbit (GEO) [8]. LEO is the lowest orbital band above the Earth's surface. This orbit offers several advantages, including lower launch costs, shorter communication delays, and faster revisit times for Earth observation satellites, making it ideal for Earth-centric missions [8].

However, nanosatellite deployments are gradually extending to higher orbital altitudes. MEO is commonly used for satellite navigation systems due to its balance between coverage and latency, while GEO is ideal for providing continuous communication services, as satellites in this orbit remain fixed relative to the Earth's surface [8].

Initially, nanosatellites were deployed in LEO for applications such as communications and Earth observation [62]. However, recent technological advancements have expanded the capabilities of nanosatellites, enabling their use in interplanetary missions. A recent example is NASA's MarCO CubeSats, which accompanied the InSight Mars lander to relay real-time data during its descent and landing [14]. This mission demonstrated the potential of small satellites for deep-space communication, opening new possibilities for low-cost exploration beyond Earth orbit.

However, all these advantages come with some limitations. The minimal structure and the typical use of COTS components, which are usually not designed specifically for space, make these systems more vulnerable to the harsh space environment. Electronic components can experience faults when exposed to radiation and cosmic rays, which are commonly present in orbit.

Another vulnerability is the limited onboard resources. Nanosatellites face significant constraints in power availability, physical space, and radiation shielding, which make traditional fault-tolerance methods, such as hardware redundancy, very difficult to implement. Moreover, radiation exposure increases with distance from the Earth's surface. At higher orbital altitudes, nanosatellites encounter a more intense radiation environment due to the absence of atmospheric shielding.

For all these reasons, it becomes crucial to study and understand how nanosatellite hardware responds to radiation-induced faults. This knowledge allows engineers to select the most reliable system architecture or implement fault-mitigation techniques to prevent system errors or, in the worst case, system crashes that could lead to mission failure.

2.2. Radiation-induced Faults

In space, electronic components are constantly exposed to high-energy radiation caused by solar events and cosmic rays. These particles interact with semiconductor materials, introducing temporary or permanent malfunctions [15]. Radiation can interfere with memory and logic circuits, leading to data corruption or unexpected system behavior. In some cases, it might affect the final output of an application, or worse, crash the system entirely [21].

The main causes and consequences of deviations from the expected function of a system are:

- **Fault**: a physical defect or imperfection in hardware or software
- **Error**: a deviation from accuracy or correctness, as a consequence of a fault
- **Failure**: the inability to perform an expected action, as a consequence of an error

When a fault causes an incorrect change in a machine stage, an error occurs. Although a fault remains localized in the affected code or circuitry, multiple errors can originate and propagate throughout the system. When the fault-tolerance mechanisms detect an error, they may initiate several actions to handle the faults and contain their errors. Otherwise, the system eventually malfunctions and a failure occurs [76].

Radiation can strike hardware components and cause radiation-induced faults [48], which can be classified based on their persistence into *permanent faults* and *transient faults*, depending on whether their effects are permanent or temporary [39].

2.2.1. Permanent Faults

Permanent faults are caused by irreversible damage to electronic components, such as semiconductor junctions, due to thermal aging, manufacturing defects, or misuse. Recovery is only possible by repairing or replacing the damaged component or subsystem [76], which is not always feasible for nanosatellites due to their large distance from Earth.

These faults are typically the result of intense radiation exposure, which can cause irreversible physical degradation in memory cells or logic circuits, introducing permanent hardware damage that may affect the functionality of the system.

One common example is the stuck-at fault, in which a memory bit becomes fixed at a constant value, either 0 or 1, regardless of write operations. When a stuck-at fault occurs, the value of the damaged cell never changes, indicating a loss of control over the affected

cell. Consequently, the cell becomes unreliable for storage and may potentially corrupt program execution or data integrity.

2.2.2. Transient Faults

Transient faults are triggered by external environmental conditions, such as power-line fluctuations, electromagnetic interference, or radiation. They usually do not cause lasting damage to the affected component, but can temporarily induce erroneous states in the system [76].

Transient faults occur when radiation temporarily alters the state of a circuit without causing permanent damage, and can be further divided into several categories, including:

- Single Event Upset (SEU) [61]: a charged particle may invert the state of a memory bit without causing physical damage. The hardware remains structurally intact, but the bit flip can affect computations or program flow and may propagate into higher-level logic, impacting results or control decisions. If the bit flip affects a single isolated cell, it is called a Single Bit Upset (SBU); if it affects contiguous cells, it is called a Multiple Bit Upset (MBU) [60].
- Single-Event Functional Interrupt (SEFI) [41]: characterized by the temporary failure of whole functional blocks. Components such as memory controllers or processors may become unresponsive, often requiring a system reset or power cycle to restore normal operations [43]. They do not cause permanent damage, but they can disrupt critical subsystems, becoming a significant issue for space missions.

2.3. Non-volatile Memories in Space

As previously discussed, the exposure of electronic components to the space environment can lead to damage caused by radiation striking the hardware, potentially resulting in either permanent or temporary malfunctions [9]. One particularly vulnerable component is non-volatile memory, which is responsible for permanently storing data. If a charged particle strikes this device, there is a risk that the stored data will be corrupted, leading to data integrity issues and errors that could potentially compromise the overall functioning of the system.

Recently, emerging COTS non-volatile memories have been proposed as innovative and cost-effective solutions to mitigate radiation effects compared to traditional radiation-hardened alternatives [48]. However, COTS devices are not specifically designed to ensure reliable operation under such harsh conditions.

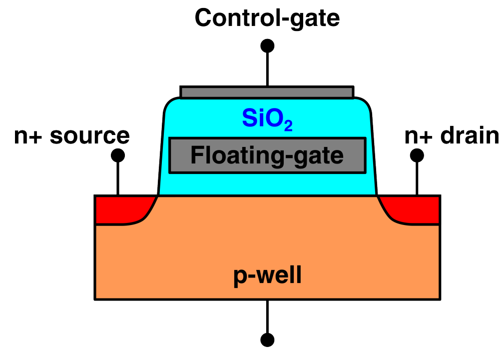


Figure 2.2: Flash memory cell schema [10].

Each type of non-volatile memory responds differently to radiation in the space environment. Radiation is a constant presence in space, so the memories used must be able to perform reliably over time, even under continuous exposure. For this reason, choosing the right memory technology is a crucial phase of designing fault-tolerant systems for space applications, to prevent data corruption and failures.

Different technologies such as Flash, FRAM, MRAM, ReRAM, and Phase-Change Memory (PCM) offer various trade-offs in terms of speed, durability, power consumption, and radiation tolerance. They are also commonly used in low-power systems, making them particularly suitable for space missions where energy resources are limited. In this section, the key characteristics of each memory technology are presented.

2.3.1. Flash Memory

Flash memory is a type of non-volatile memory that can be electrically erased and reprogrammed. The two most common types of flash memory are NOR flash and NAND flash, whose names refer to the type of logic gate used.

Despite having different architectures, both logic gates use the same fundamental memory cell structure, known as the *floating-gate MOSFET* [57], as illustrated in Figure 2.2, which shows the physical structure of a typical flash memory cell. In these cells, electrons are trapped within a floating gate that is electrically isolated by a thin oxide layer. This insulation prevents the charge from dissipating even when the device is powered off, allowing flash memory to retain data.

The flash cell acts like an electrically controlled switch, with current flowing between the source and drain terminals regulated by two gates: the control gate and the floating gate. The floating gate is positioned between the control gate and the channel, completely insulated by oxide. Because of this isolation, electrons injected into the floating gate

remain trapped, altering the device's electrical behavior [57].

When the floating gate accumulates electrons, it screens the electric field from the control gate, thereby increasing the cell's threshold voltage. This change in threshold voltage alters the drain-to-source current for a given gate voltage, enabling the cell to represent a binary data value [57].

The process of moving electrons from the control gate and into the floating gate is called *Fowler–Nordheim tunneling*, and it is a quantum tunneling process induced by applying a high electric field during programming or erasing [57]. Writing operation charges the floating gate by injecting electrons, while erasing removes electrons with a reversible and controlled operation.

A limitation of this memory type is the possible degradation that occurs due to the high electric field on the oxide, because repeated programming and erasing stress the oxide layer with high voltages, causing physical degradation over time. Such high voltage can break atomic bonds over time in the oxide, gradually degrading its electrically insulating properties and allowing electrons to pass freely from the floating gate into the oxide. This problem increases the likelihood of data loss and is why data retention decreases over time.

This effect is further amplified by the way the write operation is performed. Flash memory cells are organized into pages, which are the smallest units that can be programmed in a single operation and typically contain several kilobytes of data [31]. When writing, if we need to update only some cells within a page, the entire page is first read and copied into a temporary register. The necessary values are then modified in the buffer, and finally the entire page is programmed again, overwriting all its original content, even if only a single bit needed to be changed [31].

This approach is therefore both time-consuming and energy-intensive, because every write operation involves this full read–modify–write cycle. As a consequence, even cells that did not require changes are rewritten, which unnecessarily increases wear and accelerates their degradation.

For these reasons, flash memory is not an ideal choice for low-power devices, particularly for onboard components of nanosatellites, where energy is limited and there is no possibility of replacing a component if it degrades too quickly.

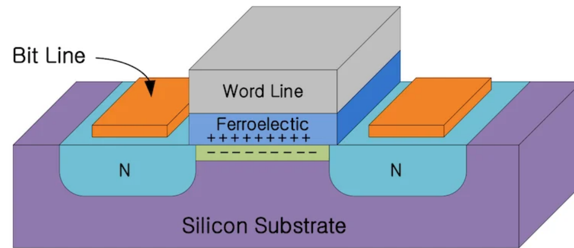


Figure 2.3: FRAM cell schema [11].

2.3.2. FRAM

FRAM is a type of non-volatile memory that stores data using a ferroelectric material, which retains electric polarization even without power. Data is stored using the polarization of tiny ferroelectric crystals. When an electric field is applied, the crystals align in one of two possible states, representing binary values 1 or 0 [43].

Figure 2.3 describes the structure of a single FRAM cell, which contains a thin film of ferroelectric material, often lead-zirconate-titanate, commonly referred to as PZT. The atoms in the PZT layer change polarity in an electric field, producing a binary switch [35]. In the writing phase, a voltage is applied to set the polarization state of the ferroelectric material within the cell, storing the data bit. However, the reading phase presents a key challenge because it is a destructive process [63]. To determine the stored value, the cell's actual polarization must be detected, but this detection alters the state of the cell itself. When reading, if the cell holds a 0, there is no significant change on the output lines. Instead, if the cell contains a 1, the shift in atomic alignment within the ferroelectric film produces a brief current pulse as electrons are displaced from the metal layer. Since this detection process disturbs the original polarization, it effectively erases the stored data during the read operation. To maintain data integrity, the cell must be rewritten immediately after reading, restoring the original polarization [63]. This requirement makes FeRAM's read operation destructive and necessitates a refresh cycle after every read.

Unlike Flash memory, where the erase operation can only be performed on entire pages, as previously explained, FRAM can write data at the bit level. This allows the user to set the memory space dynamically, making it more flexible and leading to a more efficient use of memory [63].

Several ground-based radiation experiments have been conducted to evaluate these memory technologies under harsh environments, revealing the occurrence of both SEFIs and SEUs. For instance, some experiments report FRAM tests under proton irradiation at three different energy levels, detecting both SEFIs and SEUs [54], while others describe

FRAM exposure to X-rays and protons, where stuck bits were observed without data corruption [74]. These results are important for assessing the reliability of this memory technology under radiation, but they do not necessarily represent its actual behavior in the space environment, since the particles that strike the memory in orbit may differ from those tested in the laboratory.

The main advantages and defining features of FRAMs can be summarized as follows:

- Read access time equal to write access time, both under 100 ns [43], which allows for very fast and predictable data operations, making FRAM suitable for applications that require low-latency memory access.
- Read energy equal to write energy [43], meaning that the power required to read data is the same as that for writing data, which helps to simplify power budgeting and improve overall energy efficiency in low-power devices.
- High write endurance of up to 10^4 cycles [43], allowing the memory to be rewritten many times without significant degradation, which is particularly important for applications that require frequent updates to stored data.
- Low power consumption [43], enabling FRAM to operate efficiently in battery-powered or energy-constrained systems, while maintaining high performance and reliability even in harsh environments.

2.3.3. MRAM

MRAM is a type of non-volatile random-access memory that stores data through the orientation of magnetic domains within ferromagnetic materials. These magnetic domains are microscopic regions where the magnetic moments of atoms align uniformly, creating a stable magnetic state that represents data [65]. Unlike conventional RAM, which stores data as electrical charge or current flows, MRAM encodes information magnetically, offering greater resilience to power interruptions and environmental disturbances.

As we can see in Figure 2.4, each cell consists of two ferromagnetic plates separated by a thin insulating layer, which can hold a magnetization. One of the two plates, called the *reference layer*, is a permanent magnet set to a particular polarity, while the magnetization of the other plate, called the *free layer*, can be changed to store data [67]. This configuration is repeated for each cell and is known as *Magnetic Tunnel Junction (MTJ)* [65].

The electrical resistance of the cell changes with the relative orientation of the magneti-

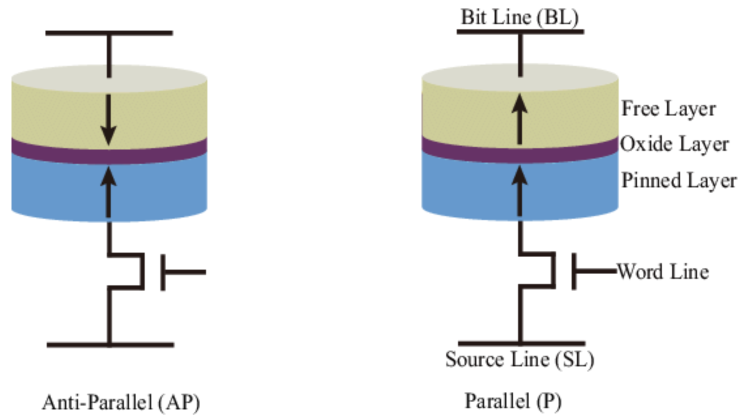


Figure 2.4: MRAM cell schema with parallel and anti-parallel configurations [75].

zation in the two plates. When the magnetizations of the reference and free layers are aligned parallel to each other, the cell exhibits a low resistance state, which is interpreted as a binary value 1. Instead, when the magnetizations are antiparallel, the resistance is higher, representing a binary value 0 [51]. Writing data involves applying a small current that changes the magnetization direction of the free layer, flipping the bit as required. Instead, the read operation is accomplished by measuring the current magnitude via the access transistor at a given gate and drain voltage.

Several ground-based radiation experiments have been conducted to evaluate the reliability of this memory technology. For instance, under certain radiation conditions, MRAM has demonstrated resilience to SEUs [38], while other studies have reported the occurrence of SEFIs in specific environments [36, 52]. Such experiments are important for assessing memory reliability and identifying potential fault mechanisms. However, their results may not capture the real behavior observed in orbit.

The main characteristics of MRAM devices are [48]:

- Low voltage [43]: they operate with standard voltage levels, which helps reduce power consumption.
- High performance [43]: they provide fast read and write access times, suitable for high-speed applications.
- Scalability [43]: they can be scaled down to smaller nodes without significant loss of performance or reliability.
- Reliability [43]: they are robust against data retention issues and can maintain stored information under challenging conditions.
- Remarkable endurance [43]: they can sustain a very high number of write cycles

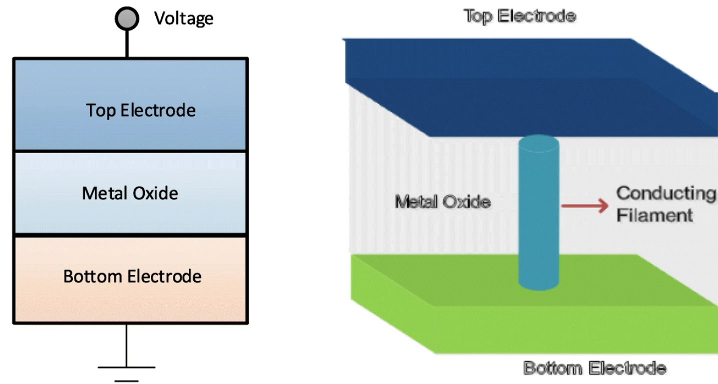


Figure 2.5: ReRAM cell schema and conducting filament creation [73].

without degradation, making it suitable for applications with frequent memory updates.

2.3.4. ReRAM

ReRAM is a type of non-volatile memory that operates by altering the electrical resistance of a dielectric material, rather than storing charge directly [28].

As illustrated in Figure 2.5, a typical ReRAM cell contains a component called a *memristor*, a contraction of memory resistor [28]. Its resistance changes from a high-resistance state to a low-resistance state depending on the voltage applied across it.

Under normal conditions, a dielectric material does not conduct electric current. Dielectric substances are used in capacitors for the specific purpose of preventing the flow of current and maintaining electric charge separation. However, if a sufficiently high voltage is applied, it can conduct because of a phenomenon called *dielectric breakdown* [27]. In a conventional dielectric material, breakdown causes permanent damage and failure of the associated component. In contrast, in a memristor, dielectric breakdown is temporary and reversible due to the materials used, allowing the device to reliably switch between states without degradation [27].

ReRAM cells typically use a resistive switching with a *metal-insulator-metal* structure [72]. Numerous substances have been tested for memristive behavior, including nickel oxide, titanium dioxide, various solid electrolytes, and other semiconductor materials.

Each cell works through the controlled creation and disruption of conductive filaments or paths within a metal oxide layer. This is possible thanks to a phenomenon called *oxygen vacancies*, which refer to missing oxygen atoms in the lattice [27]. Applying a certain voltage induces the formation of conductive filaments in the insulating layer, enabling

electron flow. This low-resistance state represents the binary value 1. Instead, when an opposite voltage is applied, the filaments disappear and the material returns to a high-resistance state, which represents the binary value 0. This process is reversible, fast, and uses little power.

Some ground experiments have been conducted on different ReRAM technologies. For example, exposing various ReRAM devices to heavy ions from different sources, SEFIs were observed, but no SEUs were detected [24, 46]. These results may not reflect the behavior of the memory in space, but they provide a valuable basis for understanding the types of faults that can occur in these memory devices.

The main advantages of ReRAM are listed below:

- Small size [43]: cells are highly compact, allowing for higher memory density and more efficient use of silicon area.
- Low power consumption [43]: it requires minimal energy for both read and write operations, making it suitable for low-power applications.
- Analog behavior [43]: it can operate in multiple intermediate resistive states, enabling analog-like memory behavior and more flexible data storage.

2.3.5. PCM

PCM is a non-volatile memory that stores data by changing the physical state of a material, usually a combination of germanium, antimony, and tellurium, between amorphous and crystalline phases. Figure 2.6 illustrates the structure of a single PCM cell, which uses two electrodes and a heater to change the state of the intermediate material. If the temperature changes rapidly from high to low, the material becomes amorphous, and its high resistance represents the binary value 0. Instead, if the heating phase is slower, the material becomes crystalline, and its low resistance represents the binary value 1. These changes are reversible and stable, so the memory keeps data even without power [48]. For writing operations, electrical pulses are used to heat the material and switch its phase, while for reading operations, we have to measure the resistance to detect the current phase.

This memory type is intrinsically radiation-resistant because it does not store charge that can be disrupted [48]. However, its behavior under different extreme conditions depends on the specific technology used to build the device. Indeed, there are many different types of PCM, and it is still in the early stages of being used for space applications [71].

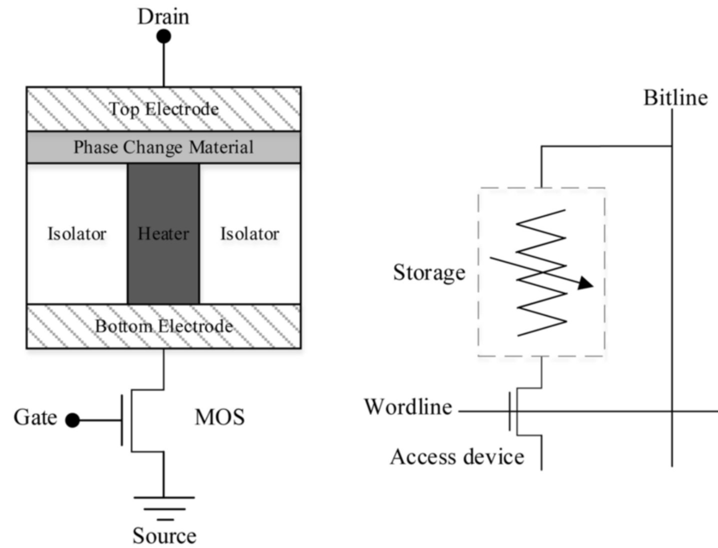


Figure 2.6: PCM cell schema and its equivalent circuit [45].

2.4. Fault Injection Techniques

After understanding how space radiation impacts electronic systems, it is important to simulate their effects on software and hardware components.

As a first step, it is necessary to define the fault models. A fault model describes the faults that the system is expected to experience during operation, depending on the nature of the system, its operational environment, the development process, and the technologies involved [50]. It requires specifying the type of fault to be introduced, the timing of the fault injection, and the part of the system targeted by the injection [50].

In this context, fault models for devices used in space are generally defined from experiments that investigate how faults occur, often relying on ground radiation tests. The main limitation is that fault rates observed during radiation testing do not necessarily match those in LEO [59]. Therefore, the fault model remains an approximation of real faults.

After the definition of the fault model, fault injection can be performed. Fault injection is a technique in which faults are intentionally introduced into a system to evaluate its behavior under fault conditions [50]. In this process, the injections directly reflect the fault model, specifying what to inject, where to inject it, and when [47].

Fault injection techniques are widely used to evaluate the dependability of systems. They involve intentionally introducing faults to observe system behavior and assess the effectiveness of fault-tolerance mechanisms. This can be achieved through physical methods, such

as radiation beams, or through software-based methods, such as altering CPU registers or memory [76].

A disadvantage is that if the fault model does not accurately represent real systems, the fault injection results may not reflect real-world behavior. This highlights the importance of performing experiments under real conditions to create fault models that are as accurate as possible.

2.4.1. Hardware Fault Injection

Hardware fault injection is a technique that introduces faults directly into the physical hardware, using tools like heavy-ion radiation, electromagnetic interference, voltage disturbances, or pin-level manipulation to simulate faults [39].

This class of techniques can be divided again into two subclasses:

- **Hardware fault injection with contact** [76]: this technique involves physically interacting with the target system, typically through direct manipulation of the hardware using tools such as probes or sockets. For example, pin-level active probes can be used to inject faults like stuck-at faults by directly altering signals at the pins of a chip. Alternatively, a socket can be placed between the hardware and its circuit board to inject faults by forcing analog signals onto the pins.

  The main advantages of this technique are its ability to provide precise control over fault injection in specific hardware locations and its capability to model complex logic faults by manipulating pin signals (e.g., inverting, ANDing, or ORing signals). However, the main disadvantages are that it is limited to faults that can be injected at the pin level, requiring physical access to the hardware, which makes it intrusive and may require briefly altering the processor's operation to inject faults.

- **Hardware fault injection without contact** [76]: in this case, faults are injected indirectly by creating physical phenomena that affect the hardware, such as radiation or electromagnetic interference. For example, heavy-ion radiation can be used to cause spurious currents inside the chip, simulating transient faults, or by using electromagnetic interference, we can alter the behavior of the hardware without direct physical contact.

  The main advantages of this technique are that we can inject faults into locations that are inaccessible by physical contact, and it is non-intrusive, as it does not require direct interaction with the hardware. Instead, the disadvantages are that it is less precise compared to contact methods, it requires specialized equipment

to generate the physical phenomena (e.g., radiation sources), and it may introduce risks of permanent damage to the hardware.

In summary, the key differences are that contact methods involve direct physical manipulation of the hardware, offering precise control but with limited accessibility and potential intrusiveness, while non-contact methods, on the other hand, rely on external phenomena to induce faults, providing access to unreachable locations, but with less precision and higher equipment requirements [76].

2.4.2. Software Fault Injection

Software fault injection is a technique to evaluate how a system reacts to faults by introducing errors into its software [76]. This method typically involves modifying the software running on the target system to simulate errors such as memory corruption, faulty disk reads, dropped or duplicated network packets, and incorrect system flags.

Faults can be injected either at compile-time, by altering the source or assembly code before execution, or at run-time, which allows dynamic interaction with the system state [39].

One of the main advantages of software fault injection is the ability to target both applications and operating systems, and it can be run in real-time, allowing to perform a large number of experiments efficiently.

This method does not require special-purpose hardware, which leads to low development and implementation costs, and it is flexible, so it can be easily extended to new types of faults. However, there are also several limitations. For example, faults can only be injected at the software-accessible level, such as processor registers and memory, which excludes deeper hardware layers [76].

2.4.3. Simulation-Based Fault Injection

Simulation-based fault injection involves the creation of a detailed simulation model of the target system. Faults are then injected into this model, and the system behavior is observed under faulty conditions [76].

An elementary fault injection experiment is defined as a single simulation run of the target system during which one or more faults may be injected at various locations within the model, and at one or multiple points in time. A series of experiments is composed of a sequence of such elementary fault injection runs, each evaluating different fault scenarios to assess the system's behavior under faulty conditions.

Some of the main approaches to perform simulation-based fault injection are:

- **Code modification** [76]: this approach is based on two elements, the *saboteurs* and the *mutants*. The firsts are additional components introduced into the model to inject faults by altering signals during the simulation; they are easy to use but limited in modeling different fault types. The seconds are modified models containing dormant fault logic blocks activated during simulation. This allows precise modeling at any abstraction level but requires replacing original components and recompiling, leading to lower performance. However, saboteurs and mutants may introduce additional simulation overhead due to increased code complexity and control logic.
- **Simulation command-based injection** [76]: it uses built-in commands of simulation tools to manipulate signals or variables at runtime. This approach offers high performance and ease of setup, especially suitable for simple fault models.

The main advantages of this approach are that it is applicable at all abstraction levels and does not require any special hardware. It is also capable of modeling both transient and permanent faults, with full control over fault types and injection timing [76]. Instead, the main disadvantage is that it requires significant development effort to build accurate models. It relies on the quality of the model: poor modeling leads to misleading results [76].

2.4.4. Hybrid Fault Injection

This combines two or more fault injection techniques to leverage their strengths while minimizing their weaknesses [76]. By integrating software-based and hardware-based methods, the hybrid approach offers a balance between flexibility, speed, and realism. Software-based fault injection allows rapid testing with minimal setup, while hardware-based techniques provide high timing accuracy and realism under actual operating conditions. For example, we can combine hardware-based fault injection for accuracy with software-based fault injection for flexibility. Moreover, hybrid methods often incorporate simulation-based fault injection to take advantage of its superior controllability and observability.

The main advantage is the reduced setup and execution time, especially for large and complex systems [76]. It combines the strengths of multiple techniques, such as flexibility, precision, speed, and realism, and it enables targeted analysis by applying the most appropriate method to each part of the system. However, this method also brings some challenges, such as the need for careful coordination between different fault injection tools and environments. It can also involve a significant integration effort, especially

when combining simulation with hardware-based methods [76].

2.5. Generative Neural Networks

If a large amount of fault-related data can be collected, such as the locations of bit flips within a non-volatile memory, it is possible to use a neural network–based approach to generate new data consistent with the observed patterns, thereby producing synthetic datasets.

This approach is particularly valuable in scenarios such as faults in onboard memories of nanosatellites, where generating synthetic data starting from a sample of the real ones can bring some advantages due to the high costs and long timeframes required to design and conduct experiments for collecting large amounts of real data in orbit.

Neural networks are the foundation of modern machine learning and can automatically recognize patterns by learning from existing examples. They are inspired by the structure of the human brain, so they are composed of layers of interconnected units that process the input data through a sequence of weighted operations and non-linear transformations [40].

Generative neural networks, unlike traditional models that only recognize or classify data, are able to generate entirely new content that closely resembles real-world data. These models try to capture the underlying distribution of the data and to generate new samples that have the same statistical properties of the training data [37]. They have gained considerable attention because of their ability to create realistic images, synthesize speech, and perform data augmentation across various domains.

Two of the most common architectures are Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), and each of these approaches offers different trade-offs in terms of training stability, sample quality, and interpretability.

2.5.1. GAN

GAN consists of two main neural network models that work together to create realistic synthetic data [29].

- **Generator Network:** the generator takes random noise as input and tries to generate realistic data samples similar to those used for the training phase. It learns the underlying data patterns by adjusting its internal parameters during training through backpropagation, and its objective is to produce samples that the discriminator classifies as real [29].

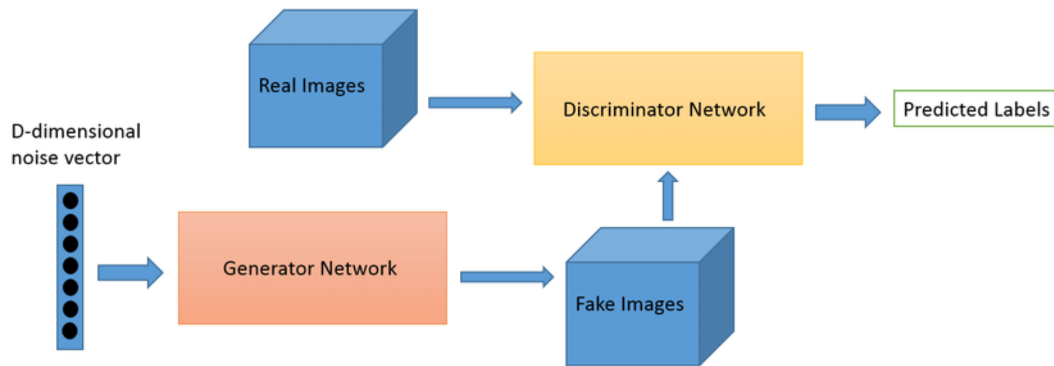


Figure 2.7: GAN training flow [29].

- **Discriminator Network:** the discriminator acts as a binary classifier that distinguishes real and generated data. It learns to improve its classification ability through training and tuning its parameters to detect fake samples with more accuracy. The goal of this network is to correctly classify input data as real or fake [29].

During the training phase, generator and discriminator networks compete and improve together, following the flow structure described in Figure 2.7 The generator takes as input a random noise vector and uses it as a starting point to create fake data samples. Its internal layers transform this noise into something that looks like real data. Then, the discriminator receives as input two types of data: real samples from the actual training dataset and fake samples created by the generator.

The job of the discriminator is to analyze each input and classify data as real or fake, so whether each sample is real data or something created by the generator. The output is a probability score between 0 and 1, which indicates the probability that the sample taken as input is real. If the generator fools the discriminator by creating realistic fake data, it receives a positive update, and the discriminator is penalized for making a wrong decision.

Through many iterations, the generator improves and creates more convincing fake samples. The discriminator also learns continuously by updating itself to better spot fake data. As training continues, the generator becomes highly capable of producing realistic data. At the end of the process, if the discriminator struggles to distinguish real data from fake, we can confirm that the GAN has reached a well-trained state. At this point, it is possible to use only the generator to produce high-quality synthetic data.

In conclusion, this model can produce high-quality results that can be used to generate highly realistic synthetic data, but it suffers from some problems which may lead to a

difficult training process, such as *model collapse*, in which the generator produces limited types of outputs repeatedly [23], and *unstable training*, in which the generator and the discriminator may not improve smoothly [29].

2.5.2. VAE

VAE is a specific autoencoder, a type of neural network designed to learn compressed representations of data [20].

The architecture of a traditional autoencoder is composed of two symmetric components: the encoder and the decoder. The encoder receives high-dimensional input data, such as an image or a sequence, and transforms it into a compact, low-dimensional representation known as the *latent vector* [20]. The purpose of this transformation is to retain only the most important features of the input while reducing redundancy or noise. In contrast, the decoder takes this latent vector and tries to reconstruct the original input. When trained properly, the decoder learns to reverse the compression performed by the encoder [20].

In this traditional model, each input is mapped to a single, fixed point in the latent space. While this works well for compression or reconstruction tasks, it poses a limitation when the goal is to generate new data because the latent space does not have a defined structure. For this reason, sampling random points often leads to incoherent or meaningless outputs. In other words, there is no guarantee that points not seen during training will decode into valid examples. This makes generative use of the latent space unreliable [20].

VAEs address these limitations by incorporating a *KL-divergence term* into their loss function, which helps in giving a structured shape to the latent space [69]. Unlike traditional autoencoders, VAEs are generative models capable of capturing the underlying distribution of the input data. In a VAE, the encoder maps input data to a well-structured latent space defined by a known probability distribution, rather than a fixed latent vector [22]. This structured latent space allows for meaningful sampling, making VAEs more suitable than standard autoencoders for data generation. The goal is to minimize the difference between a supposed distribution and the original distribution of the dataset [26].

As illustrated in Figure 2.8, VAE is based on three main components:

- **Encoder:** the encoder takes input data like images or text and learns its key features [26]. The output consists of two vectors representing the *mean* and the *variance*, which defines a range of possibilities instead of a single number, as happens using a traditional autoencoder. The goal of the encoder is to understand the input and map its main statistical structures in the latent space.

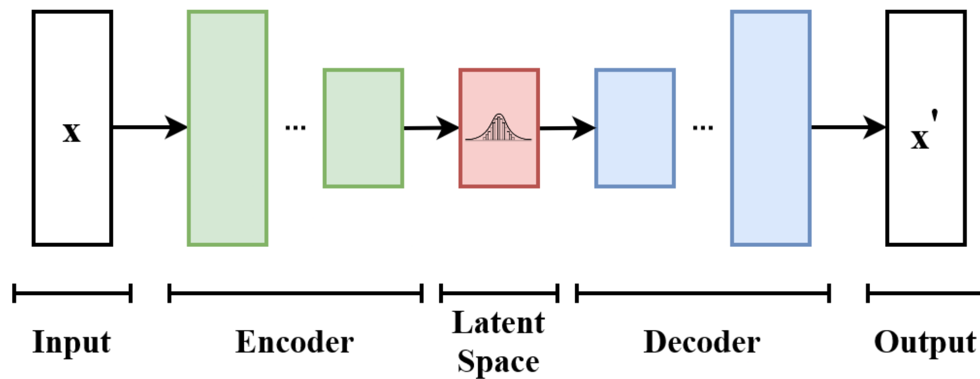


Figure 2.8: VAE training flow [45].

- **Latent space:** instead of encoding the input as one fixed point, it picks a random point within the range given by the mean and standard deviation. This process is useful to add some randomness and to let the model create slightly different versions of data. Sampling this latent space allows one to maintain the statistical structure of the real data, generating some new coherent data representation [26].
- **Decoder:** the decoder takes a random sample from the latent space and tries to reconstruct the original input. Since the encoder gives a range, the decoder can produce new data that is similar but not identical to what it has seen [26].

Once trained, the decoder can be used independently: by sampling vectors from the latent space, usually from a standard normal distribution, the decoder can generate new data points that share the underlying structure of the training data [22]. This property makes the VAE powerful in tasks such as data synthesis, interpolation, and anomaly detection.

3 | Problem Statement

In this chapter, we introduce the main problem addressed in this thesis.

Our problem is threefold:

- The first challenge is collecting data sent by the nanosatellite. This involves building a hardware infrastructure capable of transmitting data from LEO.
- The second challenge is analyzing the data received from the nanosatellite to study the faults occurring in the onboard memory devices.
- The third challenge is creating a fault model that can accurately reproduce the observed faults while maintaining their statistical properties and distribution.

Section 3.1 outlines the challenge of collecting real data to obtain realistic examples to study. Then, Section 3.2 describes the need for a method to analyze this data, to extract insights that reveal the statistical properties and distribution of the observed faults. Finally, Section 3.3 presents the challenge of creating fault models that accurately reflect the previously observed fault patterns.

3.1. Data Collection

One of the main issues is the limited availability of real-world data collected from electronic components operating in space, which are exposed to radiation and high-energy particles that can cause unpredictable faults, especially in memory components [43].

As discussed in Section 2.1, for nanosatellites, where resources are limited and redundancy is minimal, these faults represent a significant problem, potentially leading to data corruption, software malfunctions, or complete system failures.

Furthermore, as described in Section 2.4, most existing studies rely on laboratory-based radiation experiments, which attempt to emulate space radiation conditions in a controlled environment [43]. These methods have several limitations: they only approximate the intensity of particle radiation found in orbit, and they are typically performed over short

timescales, without capturing the cumulative effects of long-term exposure. There are also methods based on fault injection tools or mathematical models, but typically, the faults are only approximations, and they often overestimate or underestimate the real ones.

Because of these limitations, the resulting fault characterizations are often overly simplistic or incomplete. Without an accurate dataset derived from real orbital conditions, it becomes difficult to validate the robustness of electronic components selected for use in space. For all these reasons, we need to collect real data to understand how faults occur in memories exposed to the space environment.

Chapter 4 describes how the real experiment was implemented to expose memory devices to the LEO environment and collect data. Although this is not part of the core work of this thesis, it is included only for completeness.

3.2. Data Analysis

After collecting real data, the next challenge is to analyze it to extract insights that reveal the statistical properties and distributions of the observed faults. This analysis is important for understanding how radiation affects nanosatellite memories, how each memory technology responds to radiation-induced faults encountered in LEO, to select the most suitable memory technology for future missions, and to design more resilient devices.

An important aspect of this analysis is selecting the most appropriate metrics to evaluate memory reliability and characterize faults. For instance, it is crucial to determine how to identify regions within the memory that are more susceptible to faults, assess whether specific logical states (e.g., 0 or 1) are more prone to bit flips, and quantify faults by analyzing their main characteristics. These considerations are essential for developing a comprehensive understanding of memory behavior in orbit.

This analysis can be useful for identifying fault patterns, guiding the planning of future missions, enhancing fault tolerance strategies, and optimizing the overall reliability of onboard systems.

Chapters 5 and 6 describe the procedures used to analyze the collected data, along with a discussion of the results obtained.

3.3. Fault Models

The last challenge relates to the lack of simulation tools that can model space radiation effects based on real-world data. Existing tools for modeling and simulating space-induced faults in electronic systems, especially in non-volatile memories, are often based on ground-based experiments or generic models, as explained in Section 2.4.

These do not accurately reflect the real radiation conditions encountered in orbit. As a result, designers lack realistic data and fault models, which leads to an incorrect estimation of system vulnerability and increases the risk of mission failure in nanosatellites. There is a need for more realistic approaches that rely on actual in-orbit data to understand and predict how systems behave in the space environment.

Current simulators often employ analytical models with idealized assumptions about fault rates and distributions, synthetic data derived from laboratory testing that may not accurately represent actual space behavior, or general-purpose fault injection tools that lack device-specific knowledge.

As a result, these simulators often fail to capture important details such as the correlation between fault locations and specific memory regions, the temporal dynamics of how faults happen over time, and differences between memory technologies (e.g., FRAM, MRAM, ReRAM).

A realistic fault model should be derived from actual fault patterns observed in orbit. By using data collected and analyzed from a real satellite, it becomes possible to build models that reflect the true behavior of faults in specific memory components exposed to the space environment in LEO.

Chapter 7 describes how faults are reproduced and how they can be integrated in complete-system simulators. Then, Chapter 8 presents the methods used to evaluate these models, ensuring that statistical properties and distribution accurately reflect those observed in the real data.

4 | Experimental Setup

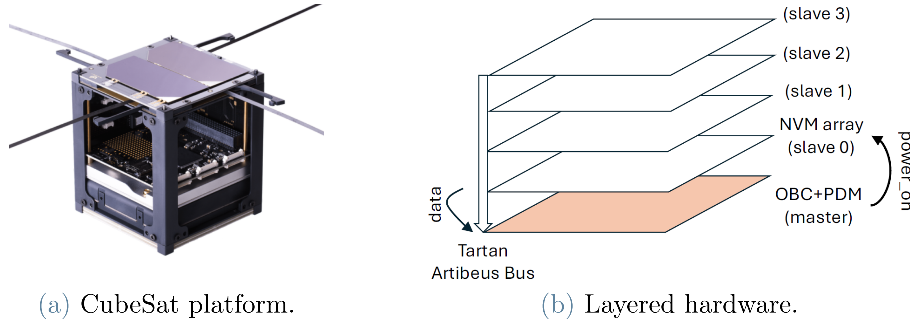
In this chapter, we describe the experimental setup, beginning with an overview of the nanosatellite's hardware architecture in Section 4.1 to provide a clearer understanding of how the system is structured, with a particular focus on the non-volatile memories selected for this objective. Then, Section 4.2 outlines the mission details, including the satellite's launch and a description of its operational behavior.

4.1. Nanosatellite Architecture

The orbital platform we deploy is the result of a collaboration involving three academic institutions and over 50 contributors throughout the design, construction, launch, and operation phases. The space vehicle, shown in Figure 4.1a, is built on the 1-Unit CubeSat platform developed by EnduroSat [1], which offers approximately 18% more internal volume than traditional CubeSat designs, while still adhering to the 1U form factor, described in Section 2.1.

The UHF Transceiver II module from EnduroSat enables downlink communication with Earth by leveraging the SatNOGS [2] global network of satellite ground stations for data retrieval. The satellite is not equipped with an active propulsion system. The onboard hardware follows a layered master-slave architecture, as schematically illustrated in Figure 4.1b. The device located at the base functions as the satellite's master On-Board Computer (OBC). It is built around a space-rated version of IBM's 6x86 CPU, a 32-bit computing core featuring a superpipelined architecture and a hardware floating-point unit. Despite being based on an almost 30-year-old design, its space-rated version is still used in satellites of various form factors, as the software written for it has undergone extensive formal verification and has proven reliable across multiple space missions [66].

A dedicated radiation-strength aluminum shield separates each of the slave devices from each other and from the OBC. Besides logging of primary mission-related parameters and general bookkeeping, the software aboard the OBC controls a custom Power Distribution Module (PDM) integrated within the OBC board. The PDM uses the energy coming from



(a) CubeSat platform.

(b) Layered hardware.

Figure 4.1: CubeSat and onboard architecture.

four CTJ30 CESI Solar cells [16] to power the OBC. These panels feature up to 29.5% efficiency and offer the largest possible effective cell area for 1U CubeSats, ultimately providing up to 2.4 W per panel in LEO. Any leftover energy is stored in a soft-reconfigurable supercapacitor array on the PDM itself, which can vary total capacitance according to the net input power. This feature is crucial in the operation of a resource-constrained nano-satellite, as input power from the solar cells may vary widely as the orbit unfolds over time.

The OBC instructs the PDM to provide power to one or more of the four slave devices onboard. These devices are programmed by independent parties to perform various experiments under the general goal of understanding the effect of the LEO environment on COTS hardware. The OBC determines what device to power among the four slaves depending on their individual energy figures and the amount of experimental data output up to a given point, in an attempt to ensure fairness of energy allocations. The slave devices relay data to the OBC through a simplified version of the Tartan Artibeus Bus [30] using a serial line.

Communication with Earth is conducted using space-proven FCC protocols and extreme data redundancy. Corrupted packets are filtered out by the SatNOGS [2] global ground station network as soon as they fail one of the multiple redundancy checks employed. This guarantees that the faults we observe occurred onboard the satellite itself and not during transmission back to Earth. Although this procedure attempts to ensure the correctness of the received data, it is not perfect, and some issues can still occur. For example, some duplicate packets might go undetected, and data with incorrect size might be received.

Due to the inherent limitations of the Tartan Artibeus Bus [30], it is not possible to accurately correlate the collected data with absolute positioning or global time, even though this information is available on the OBC. As a result, the subsequent analysis cannot make use of such temporal or positional data. It is also important to note that the

NVM model	Fujitsu FRAM MB85RS64V	Everspin MRAM MR10Q010	Fujitsu ReRAM MB85AS8MT
Bit configuration	8,192 words × 8 bits	131,072 words × 8 bits	1,048,576 words × 8 bits
Endurance	10^{12}cycles/byte	Unlimited	10^{6}cycles/4 bytes
Data retention	> 10 years	> 20 years	> 10 years
Power supply voltage (min–max)	3.3–5.5 V	1.6–3.3 V	1.6–3.6 V
Power supply current (min–max)	1.5mA–10µA	100mA–100µA	1.5mA–6µA

Table 4.1: Comparison of non-volatile memories.

data we ultimately analyze does not represent the full set generated during continuous operations, in fact, energy failures on the Tartan Artibeus Bus and data corruption during transmission reduce the amount of available data.

In our specific case, the onboard computer writes results to three different non-volatile memory devices, corresponding to the following models: Fujitsu FRAM MB85RS64V [33], Fujitsu ReRAM MB85AS8MT [34], and Everspin MRAM MR10Q010 [32].

Table 4.1 summarizes the main characteristics of each memory device, based on the information reported in their respective datasheets. In particular, it highlights the endurance (i.e., the number of read/write cycles), the minimum guaranteed data retention time, and the minimum and maximum power supply voltage and current, which provide a general indication of the energy required for operation.

We selected these three memories to evaluate different technologies, and specifically chose these models because, as explained in Section 2.3, they represent the three main types of non-volatile memories typically used in low-power devices, with high storage capacity and the ability to consume low energy for read/write operations. Their underlying technologies are also potentially well-suited for space environments due to their data storage mechanisms, which are more resistant to radiation-induced faults.

The payload includes a device executing a set of well-known algorithms, which record their outputs on the three previously selected types of non-volatile memory. These algorithms produce deterministic results. Consequently, the binary values stored in the physical memory arrays exhibit a specific and predictable bit configuration and spatial arrangement. For each of the three non-volatile memories, this results in a binary sequence that sequentially mirrors the organization of bits at the logical level.

Other technologies, such as Flash memory and PCM, were excluded due to the considerations discussed in Section 2.3. Flash memory suffers from degradation after a large number of read and write operations and has high energy consumption due to its write mechanism, while PCM is still in the early stages of qualification for long-duration space missions.

4.2. Mission

Following the selection of the non-volatile memories and the definition of the nanosatellite's architecture, the next step is the deployment of our CubeSat and the initiation of experimental operations. The satellite was launched on November, 4th 2024. It was deployed at 732 km from the Earth with an initial attitude and velocity expected to ensure roughly three months of operation. The initial estimates were far exceeded as the satellite remained operational until the end of February 2025.

When the nanosatellite accumulates sufficient power, the onboard computer begins executing the algorithms. After each execution, a memory dump is generated for each non-volatile memory and transmitted to the ground. Memory dumps serve as a virtual snapshot of the memory's logical state at a specific point in time. Concretely, they are binary sequences that sequentially represent the actual memory content onboard the nanosatellite.

On the ground, we maintain a reference binary sequence representing the expected memory state after each execution. Each memory dump received from the CubeSat is compared against this reference to detect any bit flips or other anomalies potentially induced by space radiation affecting the non-volatile memory hardware. As discussed in Section 2.4, this method represents a form of hardware fault injection without contact, induced by high-energy particles from the external environment.

When the nanosatellite is powered on and operational for our experiment, it generates a variable number of memory dumps, which are collected and asynchronously analyzed. By comparing these dumps with the reference sequence, faults can be detected and their evolution studied over the course of the mission.

In some cases, the nanosatellite may power on without transmitting any data, or it may send data processed by other onboard devices. When energy availability is insufficient, the system powers down.

5 | Analysis Framework

In this chapter, we present the procedures used for data preprocessing and fault classification, starting from the raw data collected by the nanosatellite.

Section 5.1 describes how corrupted data is discarded through data cleaning, how faults are visualized, and which fault features are selected for further analysis.

Section 5.2 outlines the methodology used to classify observed faults based on their persistence, recurrence, and reaction to system resets.

Section 5.3 discusses the method used to identify potential dependencies between faults.

Finally, Section 5.4 addresses various exceptional cases that cannot be verified during the analysis but are crucial for correctly interpreting the results and understanding the limitations of the methodology.

To better understand the following discussion, we provide the definitions of the following terms, based on the mission description of Section 4.2:

- Memory dump: the binary sequence sent by the nanosatellite that represents the state of the memory at a specific point in time.
- Reference sequence: the binary sequence that represents the correct values stored in memory and is used as a baseline to compare with the memory dumps to identify potential faults.
- Activation Session: a period during which the nanosatellite is powered on and operational for the experiment, sending memory dumps.

5.1. Data Preparation Workflow

Before running any analysis, the data received from the nanosatellites must undergo some transformations, which consist of some steps that must be executed in a sequential order.

- The first step is *data cleaning*, which involves checking and cleaning all the data to retain only the valid entries and discard any corrupted ones, if present. This step

is important because it removes data that could compromise the results of the later analysis, keeping only the reliable ones.

- The second step is *descriptive analysis*, where general metrics and other statistical indicators are computed. This provides a quantitative overview of the dataset, helping to understand the overall distribution and variability of the data before proceeding to more complex evaluations.
- The next step is *visualization*, where faults are detected and visualized to identify general patterns and distributions, if any. This provides an initial overview, allowing us to observe the shape of the faults and whether they appear grouped, scattered, or limited to specific memory regions. This is important for deciding how to analyze them in the next phases.
- The final step is *feature extraction*, which helps to determine how to identify each fault and its main characteristics. It allows us to extract the most important features of each fault, useful for subsequent analysis.

All these steps are described in detail below, explaining their implementation and highlighting their role in ensuring accurate and reliable analysis.

5.1.1. Data Cleaning

The first phase of the data preparation workflow is data cleaning, which ensures that corrupted entries do not compromise the subsequent analysis. When the solar panels collect enough energy, the nanosatellite runs its onboard algorithms and sends memory dumps to the ground. Each memory dump must have a fixed length, which depends on the total number of bits that the memory can store.

As a first check, we verify the length of each sequence to ensure that no corrupted data enters the analysis. Occasionally, sequences may be longer than expected, as discussed in Section 4.1, so it is important to check the length of all the memory dumps. If a communication error occurs, extra packets may be transmitted, causing parts of the memory dump to be repeated. The main issue is to identify where the extra packets have been inserted within the sequence.

To handle memory dumps with incorrect sizes, we compare each sequence to the reference one and truncate any oversized dumps to the expected length by removing extra bits at the end. If the number of detected bit flips in these adjusted sequences remains consistent with those in correctly sized dumps, we retain the data, as it is highly probable that the extra packets were simply appended at the end. Otherwise, we discard the sequences,

since this suggests that the extra packets were inserted in the middle of the dump. In such cases, the bit shift would misalign the data, producing many false positives that do not correspond to real memory faults.

Memory dumps with these characteristics were observed, and the results of this initial data-cleaning step are presented in Chapter 6.

5.1.2. Descriptive Analysis

After the cleaning phase, the retained data is ready for analysis. Computing basic statistics as a first step is important to gain an initial understanding of the dataset, identify general trends, and detect anomalies or irregularities.
The following metrics are considered:

- Evolution of Bit Flips: this metric shows how the number of bit flips varies across memory dumps which are sequentially ordered with their arriving from the nanosatellite. It helps identify periods of increased or decreased memory corruption, detect possible phases in fault behavior, and highlight anomalies or transient events during the mission.

  Formally, for the j-th memory dump, we define:

  $$y_j = \sum_{k=1}^{m} b_{j,k} \tag{5.1}$$

  where:

  - y_j: total number of bit flips in the j-th memory dump,
  - m: number of bits in a memory dump,
  - $b_{j,k}$: binary indicator (1 if bit k is flipped in dump j, 0 otherwise).

  The obtained sequence $\{y_j\}$ represents the evolution of the number of bit flips as new memory dumps are received.

- Mean: the average number of bit flips per memory dump provides a global view of the fault rate. It allows for a high-level comparison of the overall reliability of different memories and helps quantify the typical corruption level in orbit.
  It is defined as:

  $$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{5.2}$$

where:

- n: the total number of memory dumps,
- x_i: the number of bit flips in the i-th memory dump.

- STANDARD DEVIATION: it quantifies how much the bit flip count deviates from the mean across memory dumps. It provides insight into the consistency of fault occurrence, indicating whether faults are uniformly distributed or exhibit significant fluctuations.

  It is defined as:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{5.3}$$

  where:

  - n: the total number of memory dumps,
  - x_i: the number of bit flips in the i-th memory dump,
  - μ: the mean number of bit flips per memory dump, as defined in Equation 5.2.

5.1.3. Visualization

After selecting only the valid memory dumps and computing basic statistics, we proceed to identify the positions of the bit flips to study the distribution of faults. To this end, we introduce the following definition:

BIT-FLIP MASK: a binary sequence of the same size as the memory dump, where a value of 1 indicates a bit flip and a value of 0 indicates that the stored bit is correct.

To create bit-flip masks, we compare each memory dump to the reference sequence, performing a logical XOR operation. The result shows where the bit flips happened, as described in Figure 5.1.

After generating all the masks, we can visualize them to understand how the bit flips are distributed across the memory. For example, we may observe that flipped bits do not occur randomly but instead are often grouped in clusters or blocks. Visualization makes it easier to identify trends and to guide the next step of the analysis.

In our case, the visualization phase allowed us to observe that the bit flips are sometimes isolated and sometimes consecutive, as also reported in ground experiments on other memory models of the same type [43].

REFERENCE SEQUENCE	1	1	0	0	1	•••
MEMORY DUMP	0	1	0	1	0	•••
BIT FLIP MASK	1	0	0	1	1	•••

Figure 5.1: Bit-flip mask creation.

These observations, whose results are presented in Section 6.3, form the basis for the subsequent steps and the decisions on how to analyze the faults, which are discussed in the following sections.

5.1.4. Features Extraction

The next step of data preprocessing phase focuses on extracting detailed characteristics from the created masks. Each mask shows us where bit flips have occurred, but to make sense of this information, we need to identify patterns and quantify them.

After the visualization phase, we observe how bit flips are spatially organized within the memory array, to define the most suitable strategy for analyzing them. This decision is made based on the actual visual inspection of the bit flip distribution, shown next in Section 6.2.

Observing the visualized bit flips, we noted that they are not localized in specific memory regions. They appear sparsely distributed and can involve consecutive cells in the horizontal direction, as also observed in other studies conducted with ground-based experiments [43]. For this reason, we define the following term:

BLOCK: a sequence of bit flips of size from 1 to N that involve consecutive memory cells horizontally. A block occurs consistently at the same memory location, always affecting the same bits, that is, the same memory cells.

Given a block, in addition to its size, other important characteristics to consider are:

- DURATION: the number of consecutive memory dump in which the same block appears. A block may appear in a memory dump and then disappear after some memory dumps in the same session, disappear after the nanosatellite powers off, or persist until the end of the mission. Each time a block appears, its presence is monitored in subsequent memory dumps, and its duration is recorded. Duration

is important because it helps distinguish between transient and permanent faults, allowing the analysis of the severity of the fault.

- Occurrences: the total number of times the same block appears throughout the mission. Knowing how often a fault recurs helps estimate its frequency, indicating whether certain faults are common or rare.

- Bit flips transitions: it indicates whether, inside the block, the bits are flipped from 0 to 1 or from 1 to 0. This helps identify if one logical state is more vulnerable than the other to radiation-induced faults, which can guide the following memory design and fault tolerance techniques.

5.2. Fault Classification Procedures

As introduced in Section 2.2, memory faults can be classified based on their persistence. In particular, we distinguish between transient faults, which are temporary, and permanent faults, which suggest permanent damage. Starting from the previously extracted features, such as occurrences, size, and duration, we examine the pattern of each block to determine its category. We classify each observed block as a fault, categorize as follows:

- Permanent block: permanent fault that appears at a specific point in time and continues to be present across multiple activation sessions, even after the reset of the nanosatellite. These faults indicate radiation-induced physical damage in the involved memory cells.

- Transient block: a transient fault that appears and disappears after some memory dump, indicating a fault that can affect some bits of the memory without causing permanent damage to the hardware. These can be further categorized into:

  - SEU: transient fault that appears and disappears within the same activation session. This fault is temporary and is caused by a single radiation event affecting the memory, which then resolves without external interventions. Understanding transient faults is important because it helps identify recoverable faults caused by radiation. This can be useful to implement fault tolerance techniques and to improve reliability without unnecessary hardware replacement.

  - SEFI: transient fault that appears and persists until the end of the activation session, but disappears after the satellite powers off and restarts. This fault requires a system reset to be cleared. In our context, the reset occurs auto-

matically when the nanosatellite powers down due to insufficient solar energy and then restarts in a later activation session. Recognizing SEFIs is crucial, as it can cause system interruptions and require a reset to clear. This insight can guide the design of effective recovery mechanisms, such as automatic resets.

5.3. Dependency Between Faults

Another interesting observation is the possibility of identifying correlations between blocks by analyzing the probability that the appearance of one block is associated with the appearance of another.

The method that we used to identify dependencies between the appearances of blocks is the calculation of conditional probabilities using Bayes' Theorem. The probability of an event A given that another event B has occurred is:

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)} \tag{5.4}$$

In our context:

- A is the event that the first block appears.
- B is the event that the second block appears.
- $P(A)$ is the probability of observing block A in any memory dump.
- $P(B)$ is the probability of observing block B.
- $P(B \mid A)$ is the probability of observing B given that A has occurred.
- $P(A \mid B)$ is the conditional probability of A given B. This tells us if A is more likely when B has already appeared.

To apply this method, we collect empirical data from masks:

- Count how many times each block appears: $\#A$, $\#B$.
- Count how many times both blocks appear together: $\#(A \cap B)$.
- Compute empirical probabilities:

$$P(A) = \frac{\#A}{N}, \quad P(B) = \frac{\#B}{N}, \quad P(B \mid A) = \frac{\#(A \cap B)}{\#A}$$

  where N is the total number of bit flip masks analyzed.

Substituting into Bayes' Theorem, the results can be interpreted as following:

- If $P(A \mid B) > P(A)$, then the occurrence of B increases the likelihood of A, suggesting a positive dependency.
- If $P(A \mid B) \approx P(A)$, then the events are likely independent.
- If $P(A \mid B) < P(A)$, then B may negatively correlate with or suppress the appearance of A.

These values are computed for all combinations of observed blocks to search for correlations with all the other blocks that appeared during the entire mission.

5.4. Threats to Validity

To properly understand the following analysis, it is important to acknowledge that there are many exceptional cases that we cannot verify because they may not be visible in the analysis, even though they could still happen in reality. Furthermore, some implementation choices made during the experiment limit our ability to observe certain specific aspects.

The main issue is that there is no absolute concept of time. We do not have access to the exact timestamps of when memory dumps are created or received. Although we know the day of the reception, we lack precise information about the timing and correlation between individual dumps. The only available reference is the sequence in which memory dumps arrive, which allows us to order them sequentially but does not reveal the actual time intervals between them. Additionally, when the nanosatellite powers off, we cannot determine the exact duration of the shutdown period. This limitation means that all time-related analyses must rely on relative ordering rather than absolute timing.

Since the nanosatellite powers off periodically, the analysis focuses exclusively on faults occurring while it is powered on and able to execute programs. This does not mean faults cannot happen during power-off periods. Memories can still be affected by radiation at any time. However, we cannot study those faults because the nanosatellite only transmits data during its activation sessions.

When observing faults during the activation sessions, further challenges arise in detecting all of them. One such event we cannot observe is when a bit flip occurs at a specific position in a memory dump, and another flip happens at the same position in the next memory dump. When computing the bit flip mask, this event will not be detected because the second flip reverts the bit to its original correct value, making it indistinguishable from

a non-flipped bit.

Another issue concerns the duration of a fault. When we observe faults persisting across consecutive memory dumps, we cannot know whether this persistence is due to repeated bit flips at the same coordinates or due to the same bit flip truly persisting over time.

A further challenge arises from the inability to determine the physical arrangement of memory cells at the hardware level. The bit flips we observe are based on the logical representation of the memory, so the binary sequence in the memory dumps. Therefore, when bit flips appear in consecutive logical positions, we cannot assume the corresponding cells are physically adjacent in hardware. On the other hand, bit flips distant in the logical representation might actually correspond to physically adjacent memory cells. As a result, we cannot correlate logical fault patterns directly with their physical locations in the memory hardware.

6 | Results and Discussion

In this chapter, we discuss the results obtained using the analysis framework described in Chapter 5. The main topics addressed in this analysis are briefly summarized below, along with the corresponding sections where each aspect is discussed.

Section 6.1 highlights that a portion of the data sent by the nanosatellite has incorrect sizes and explains the criteria used to decide which data to keep or discard, to avoid compromising the results of subsequent analyses.

Section 6.2 outlines the metrics used to evaluate the overall memory corruption:

- Evolution of bit flips: to track how the number of bit flips changes as new memory dumps are received.
- Mean and standard deviation: to quantify the extent of corruption in each memory and assess their reliability.

Section 6.3 highlights the presence of blocks in all bit flip masks, explores possible dependencies among them, and classifies them as permanent or transient faults. The main steps include:

- Bit-flip masks visualization: to observe the distribution of bit flips within each memory dump.
- Block dependency analysis: to assess whether the presence of one block increases the likelihood of another appearing.
- Permanent faults analysis: to study typical sizes and stuck-at values of affected bits.
- Transient faults analysis: to examine the distributions of size and duration, the most frequent bit flip transitions, and the recurrence of each block.
- Heatmap creation: to visualize the overall bit flip distribution in each memory

Section 6.4 summarizes the previously presented results and compares the memory devices, discussing the advantages and disadvantages of each based on the findings from this experiment.

6.1. Data Cleaning Results

As explained in Section 5.1, the first step of the data preparation workflow is data cleaning. Each binary sequence representing a memory dump must have a specific length, which corresponds to the maximum number of bits that each memory can store, as illustrated in Table 4.1.

However, as discussed in Section 4.1, the packet filtering system is not entirely reliable, so additional checks on the data integrity are necessary. In practice, some memory dumps were received with incorrect lengths, containing extra bits within the sequence. These anomalies result from errors during the communication phase, where packets of information are occasionally duplicated.

The main issue with memory dumps of incorrect size is identifying the exact location of the duplicated values within the sequence. If the extra data appears in the middle, it causes a shift in all subsequent bits. This misalignment can lead to valid values appearing corrupted, even if the original memory contents were correct. To avoid including such misleading data in the analysis, these sequences must be discarded.

To determine whether the extra bits were simply added at the end of the sequence, we apply a truncation method: we cut off the extra bits at the end of the sequence and compare the result to the reference sequence. If the number of bit flips is similar to those observed in correctly-sized dumps, we can assume that the extra bits were added at the end. However, if the corruption rate is significantly higher, it suggests that the extra data was inserted in the middle, shifting the entire sequence and making it unreliable. In such cases, the dump is discarded due to the impossibility of determining the correct alignment.

In our analysis, we chose to discard all memory dumps with incorrect sizes because more than 30% of the truncated bit flip masks exhibited an anomalous number of bit flips. This indicates a high level of corruption caused by internal misalignment.

Table 6.1 presents the total number of memory dumps received for each memory type, along with the number of valid dumps and those discarded. FRAM received more memory dumps than MRAM and ReRAM. This occurs because, during the initial part of the mission, in several activation sessions, only data related to the FRAM memory was received, while no data related to MRAM and ReRAM was available.

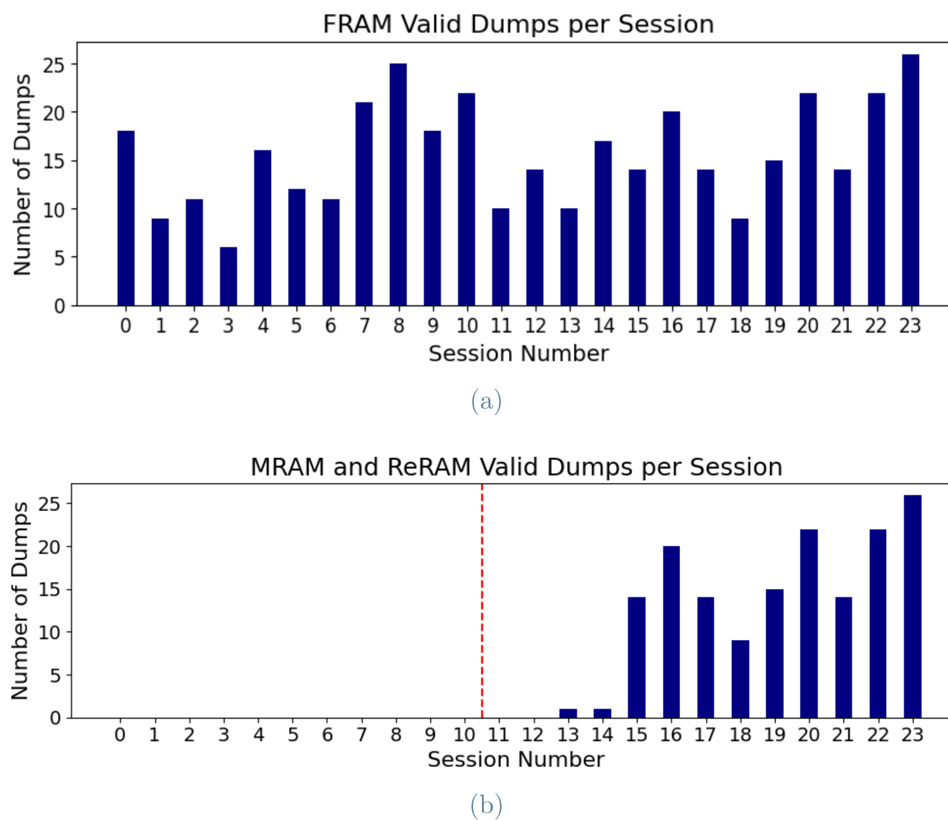
Figure 6.1a displays the number of valid memory dumps received for each activation session in the case of FRAM. In contrast, the plots for MRAM and ReRAM are identical,

Memory Type	Total Number	Invalid	Valid
FRAM	384	8	376
MRAM	211	53	158
ReREAM	211	53	158

Table 6.1: Number of valid memory dumps for each memory

as shown in Figure 6.1b. In these cases, both the number of memory dumps and the number of sessions are lower. As previously explained, this reduction is due to the absence of received data from MRAM and ReRAM during the first 11 sessions, indicated by the vertical red line in the plot.

Starting from the 12th session, data from both MRAM and ReRAM becomes available. However, all memory dumps from the 12th and 13th sessions for these two memories are discarded, as their sizes do not match the expected specifications. From the 14th session onward, a larger number of correctly sized memory dumps are received.



(a)

(b)

Figure 6.1: Valid memory dumps per sessions for each memory.

6.2. Descriptive Analysis Results

As described in Section 5.1, we first assess memory reliability by computing general metrics, including the evolution of bit flips and the mean and standard deviation for each memory type. The following sections present the results of this first descriptive analysis along with a detailed discussion.

6.2.1. Evolution of Bit Flips

As a first step, we count the number of bit flips in each memory dump over time to calculate their percentage throughout the different activation sessions. We order the memory dumps according to their arrival sequence from the nanosatellite. By comparing each memory dump with the reference sequence, it is possible to count the bit flips and compute a corruption percentage for each dump, obtained by dividing the number of flips by the total number of bits in the corresponding memory.

Figure 6.2a shows the evolution of the bit-flip percentage in the FRAM memory. Vertical red lines indicate the end of a session, highlighting the changes in the corruption percentages during each activation phase of the nanosatellite.

As shown in the plot, the trend of bit flips across the mission is divided into four distinct phases, characterized by an overall decreasing but oscillating number of bit flips. A more detailed analysis of the bit-flip distribution across these phases is presented in the following section, after the visualization step used to observe the fault distribution.

Figure 6.2b illustrates, using the same conventions as the previously described plot, the evolution of bit flips in the MRAM memory. We can observe that, during the initial period in which data are received, the memory often remains uncorrupted, although occasional peaks appear, indicating the presence of a small number of bit flips that quickly disappear.

Between the 70th and 80th memory dumps, we observe that the percentage of bit flips increases and no longer returns to zero. This suggests that an event occurred, causing the memory to remain partially corrupted. Peaks of bit flips continue to occur, but the overall level of corruption never fully disappears. We explore this observation in more detail next, investigating the possible presence of a permanent hardware fault.

Finally, Figure 6.2c shows the evolution of bit flips for the ReRAM memory. The trend is oscillating, indicating the presence of bit flips that appear and disappear within the same sessions. In the last session, the number of bit flips appears to be higher, but no other particular pattern is observed.

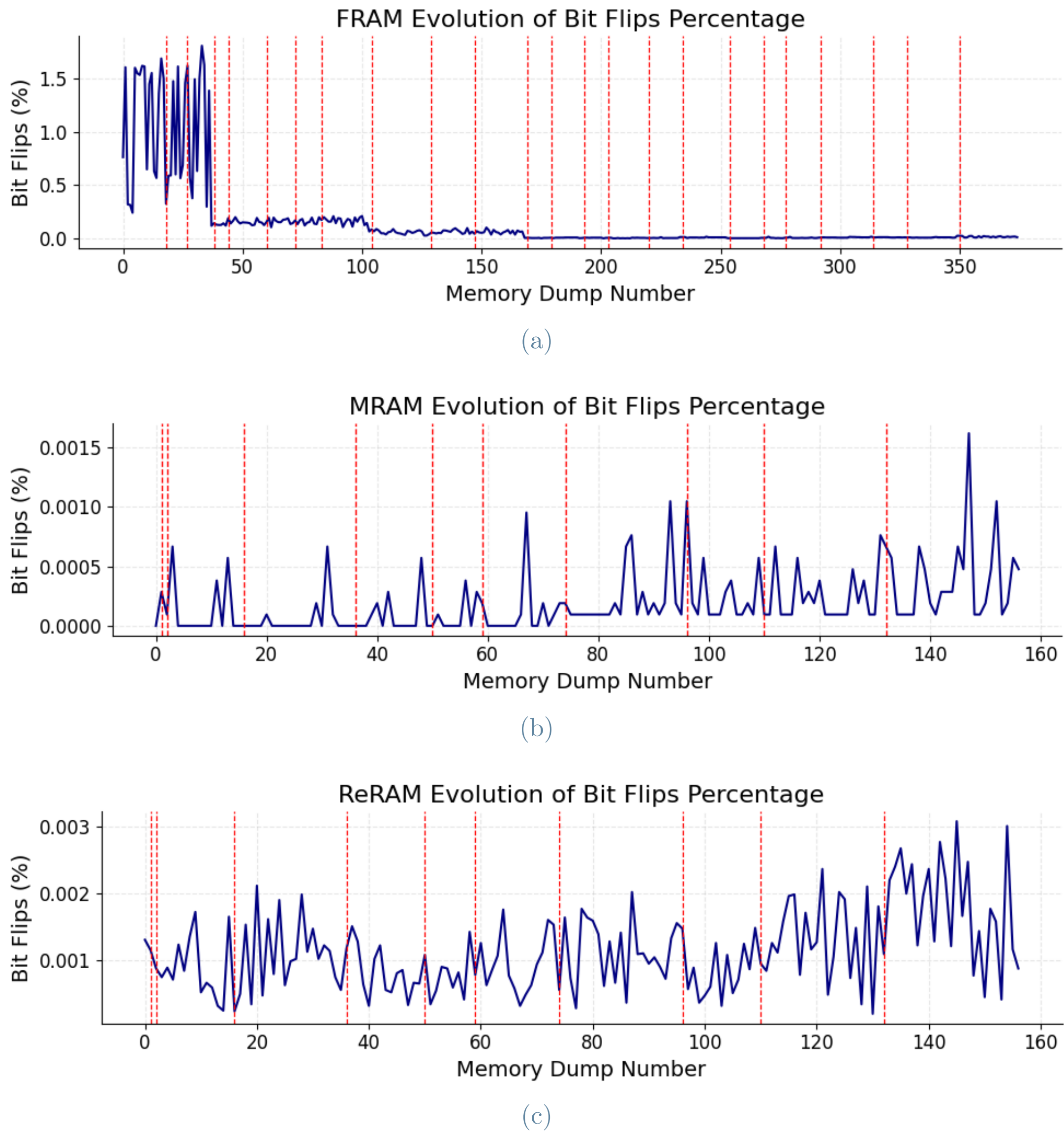


Figure 6.2: Evolution of bit flips percentage across different memory technologies.

6.2.2. Mean and Standard Deviation

As previously discussed in Section 5.1, other important metrics to calculate are the mean of bit flips, which reflects the typical number of faults found in a memory dump, and the standard deviation, which quantifies the variability of bit flip counts around that mean.

As shown in the Figure 6.2b and Figure 6.2c, both MRAM and ReRAM exhibit a relatively stable trend, allowing for the calculation of a single mean and standard deviation for each. In contrast, FRAM shows distinct phases, each with its own statistical characteristics. Based on the different corruption phases identified in Figure 6.2a, we first calculate the mean and standard deviation for FRAM across all activation sessions, and then separately

Mem. Type	Bits	Phase	Sessions	Mean Corr. Bits	Std Dev	Mean (%)
FRAM	65536	All	0-23	96.05	232.08	0.15%
		I	0-2	671.56	366.62	1,02%
		II	3-7	102.15	19.80	0,16%
		III	8-10	39.46	11.97	0,06%
		IV	11-23	2.76	2.53	0,004%
MRAM	1048576	-	13-23	1.52	2.64	0,00014%
ReRAM	8388608	-	13-23	95.27	51.13	0,0011%

Table 6.2: Mean and standard deviation of bit flips per memory type and sessions.

for each identified phase. Table 6.2 summarizes the results for all memory types, including the different corruption phases observed in FRAM.

Observing the mean absolute values, we note that for FRAM the corruption rate is highest in the first phase, drops in the second, decreases further in the third, and stabilizes in the final phase, which coincides with the period when data from the other memories also become available. The same trend is observed in the standard deviation: in the first phase it is more than half of the mean, indicating high variability in corruption rates. Fluctuations decrease in the later phases along with the mean, showing progressively more stable behavior that becomes steady in the last phase.

Considering the absolute means and standard deviations only during the period in which data from all memories are available, FRAM and MRAM show similar absolute values. However, when the mean is expressed as a percentage, MRAM exhibits lower corruption due to its larger storage capacity. In contrast, ReRAM shows the highest absolute mean and standard deviation, indicating greater fluctuations in corruption rates and a less stable behavior compared to the other two memories. Nevertheless, its mean corruption percentage remains lower than that of FRAM due to its significantly larger memory capacity, while still being higher than that of MRAM.

6.3. Features Extraction and Classification

Another key step in the data analysis is to identify the distribution of faults, examine their main characteristics, and classify them as either transient or permanent, as discussed in Section 5.2.

The first step involves visualizing the bit flip masks to observe the typical distribution of bit flips. We then analyze whether any correlation exists between them, understanding whether the presence of one fault influences the simultaneous appearance of another.

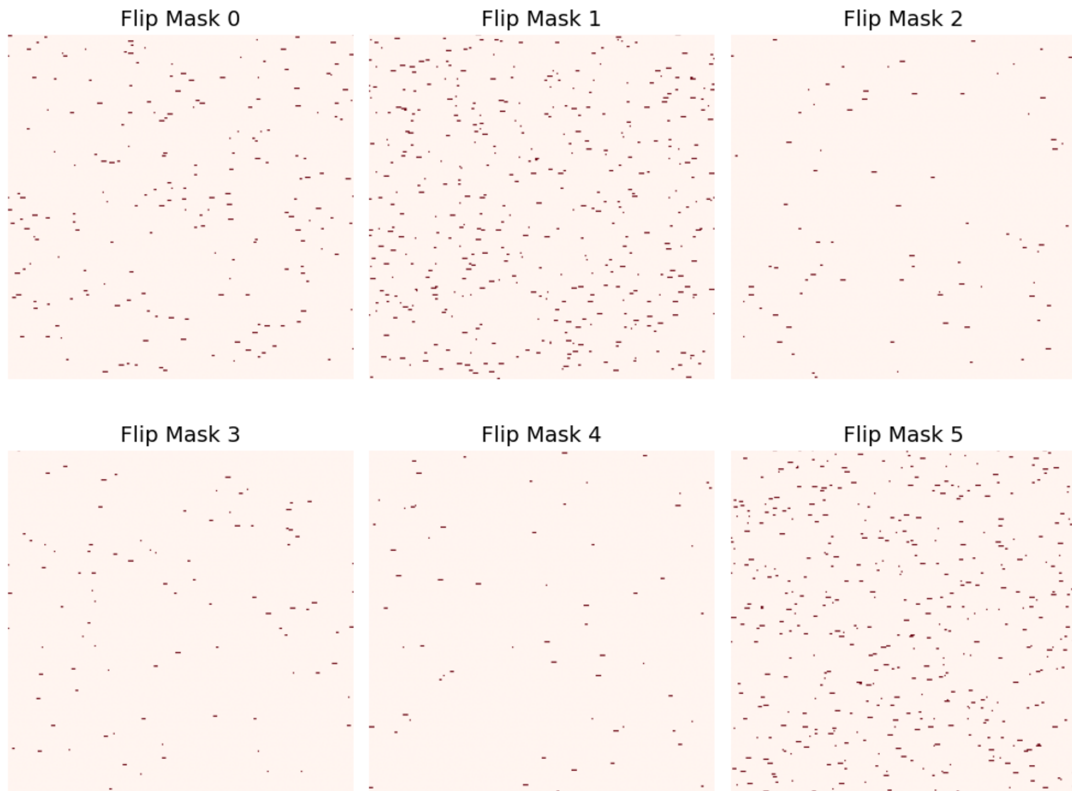


Figure 6.3: FRAM bit-flip masks examples.

Then, we perform a classification between transient and permanent faults to assess the resilience and vulnerabilities of each memory. Finally, we create heatmaps to visually determine whether certain memory regions are more vulnerable than others.

6.3.1. Fault Distribution

After analyzing and discussing the general corruption statistics of the memories, we now focus on the distribution of faults. We achieve this by visualizing the bit-flip masks, providing a graphical representation of how faults are distributed.

The approach involves reshaping the binary sequences representing the bit-flip mask into a square or rectangular matrix, filling it row by row from the top-left to the bottom-right, allowing for a clearer graphical representation. Figure 6.3 displays the first 6 bit flip masks of the FRAM memory, marking in red the 1s.

We observe sequences of consecutive bit flips, but no specific regions consistently exhibit a high density of faults. Instead, bit flips tend to appear in horizontally aligned groups of adjacent bits, apparently randomly distributed across the memory. These groups are referred to as *blocks*, as explained in Section 5.1. This pattern is consistent across all bit flip

masks, including those from MRAM and ReRAM. However, these are not shown here due to space constraints and reduced readability in the figure. Moreover, this pattern aligns with observations from ground-based experiments on the same memory technologies [43].

As shown in Figure 6.2a and Table 6.2, FRAM exhibits distinct fault dynamics during the first part of the mission, up to the 11th activation session. In the second part, when data from the other two memories becomes available, its behavior aligns more closely with the others. Analyzing the bit flip masks of the FRAM memory during the first three phases, we observe a regular spatial pattern: the distance between consecutive blocks is always a multiple of 8. This pattern disappears after the 168th memory dump, marking the end of the 11th activation session, after which no regular spacing is observed and the fault rate stabilizes. At the same time, data from MRAM and ReRAM begin arriving; however, their bit-flip masks do not show any specific spatial pattern or regular spacing.

A plausible hypothesis is that, during the first part of the mission, a malfunction in the FRAM memory controller, responsible for managing data flow between the CPU and memory, caused it to write incorrect values at regular intervals, producing blocks spaced by multiples of 8 cells. Once the controller resumed normal operation, no further spatial dependencies were observed.

6.3.2. Block Dependency Analysis

After observing the fault distribution, and once the blocks have been identified, the next step is to investigate whether these blocks exhibit dependencies among each other. To achieve this, we applied the method described in Section 5.3, which calculates the conditional probability of a block appearing given the presence of another block in the same memory dump.

For all the memories, the results are similar: the majority of the computed conditional probabilities are equal to 0 or 1, which statistically means that the corresponding faults never or always appear together.

The main challenge with this result is that most blocks appear only once and vanish immediately after a single memory dump, without reappearing during the mission. Consequently, two blocks occurring in the same dump may simply coincide by chance, making such simultaneous appearances statistically insignificant. A single co-occurrence without repetition is insufficient to establish whether the blocks are dependent or independent. The same logic applies when the observed conditional probability is zero. The fact that two blocks have never appeared together does not guarantee they never will.

Due to the limited amount of available data, we cannot draw definitive conclusions about real statistical dependencies between blocks. A larger number of observations is necessary but, unfortunately, the available data is insufficient for this purpose. However, based on our observations and analyses, we can assume that all blocks found in the memory dumps are independent from each other.

6.3.3. Permanent Faults Analysis

After identifying the presence of independent blocks, the next step is to determine whether they correspond to transient or permanent faults.

As a first observation, we noted that each memory contains a small number of blocks exhibiting a distinct pattern. Once these blocks appear, they persist across all subsequent bit flip masks without disappearing, even across different sessions. Following the classification procedure outlined in Section 5.2, we categorize these as permanent blocks because this persistent behavior suggests physical damage to the underlying hardware, resulting in a permanent fault.

The following plots illustrate, for each memory type, the cumulative number of permanent blocks detected across the activation sessions. For each session, the number of newly detected permanent blocks is represented as a step on a blue line. The horizontal axis indicates the sequential session number, while the vertical axis shows the cumulative count of permanent blocks.

Figure 6.4a illustrates the appearance of permanent blocks in the FRAM memory. We can observe the appearance of a single permanent block in two separate sessions. Later, near the end of the mission, three additional permanent blocks appear during a single session. Permanent faults are also observed in the other two memory types. Figure 6.4b shows a single permanent block in the MRAM memory. In the case of ReRAM, shown in Figure 6.4c, two permanent blocks appear simultaneously within the same session near the end of the mission.

Table 6.3 summarizes the main characteristics of all permanent blocks observed across the studied memories. It reports the size of each block, the stuck-at values indicating the logic state for each affected bit, and the number of memory dumps in which the permanent blocks appear.

This analysis shows that the most permanent faults are blocks of size 1, with bits stuck at the logic value 1. Additional observations include the presence of a single bit stuck at 0 in the FRAM memory and a block of size 2 in the ReRAM.

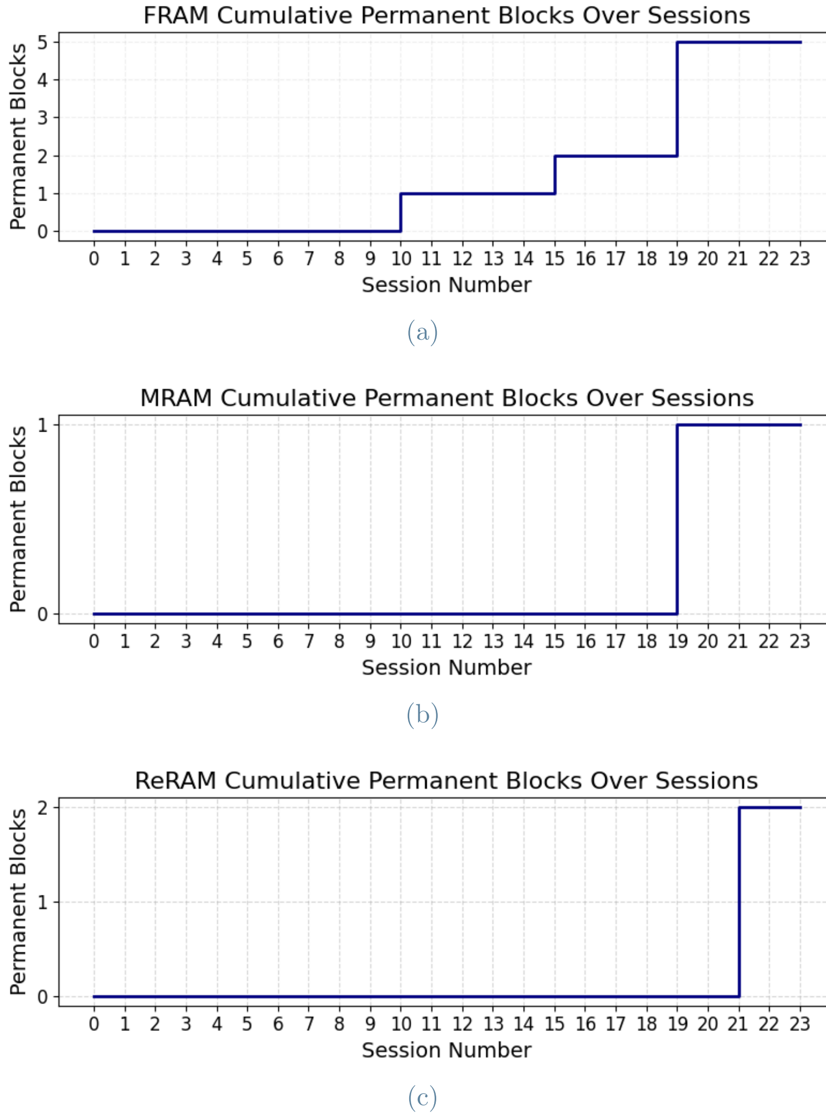


Figure 6.4: Cumulative number of permanent blocks over activation sessions.

This particular behavior leads us to formulate several observations related to the hardware structure of each memory cell, as described in Section 2.3, and to compare our results with ground-based evaluations of FRAM, MRAM, and ReRAM technologies [43].

In our experiment, we observe that in the space environment, particularly in LEO, FRAM cells are more likely to have their ferroelectric material change polarization under the influence of high-energy radiation, causing the cell to become stuck at a value of 1. This phenomenon was not observed in ground-based tests [43], where no stuck-at faults occurred. For MRAM, only a single permanent bit flip was observed, making it difficult to draw statistically significant conclusions. However, it is plausible that the magnetization of the free and reference layers in the MTJ became parallel due to radiation-induced

Memory	Size	# Dumps Involved	Stuck-at
FRAM	1	207	[1]
	1	142	[0]
	1	84	[1]
	1	84	[1]
	1	84	[1]
MRAM	1	84	[1]
ReRAM	2	48	[1,1]
	1	48	[1]

Table 6.3: Permanent blocks features for each memory.

effects, resulting in a permanent low-resistance state corresponding to a logic 1. Once again, ground-based results [43] did not report any stuck-at faults. Similarly, for ReRAM cells, the observed permanent faults suggest that the dielectric material may undergo a permanent change in resistance under radiation, resulting in a stuck-at-1 fault. In contrast, ground-based experiments [43] observed stuck-at faults at logic state 0, opposite to those detected in orbit.

These results confirm that outcomes from radiation tests conducted on the ground do not always faithfully reflect the behavior observed in actual orbital conditions.

6.3.4. Transient Faults Analysis

Excluding permanent faults, all remaining blocks are classified as transient faults due to their key characteristic of disappearing before the end of the mission. For each block, we calculate its size, duration, occurrences, and bit-flip directions, as explained in Section 5.1, and classify it as SEU or SEFI according to the procedure defined in Section 5.2.

The analysis of occurrences shows that most blocks appear only once throughout the mission, indicating that bit flips typically affect different memory coordinates each time, with a very low chance of repeated faults at the same location. Regarding bit flip directions, no clear preference is observed between logical states, suggesting that both are equally susceptible to transient faults. The other results are described below, with particular attention to the distributions of size and duration, followed by a subsequent classification of the faults.

Block Size. Each block has a specific size, indicating the number of flipped bits it contains. We analyze the size distribution across all activation sessions for each memory type to understand how block sizes vary and to identify which sizes are most common.

We begin by analyzing the distribution of block sizes in the FRAM memory. Figure 6.5a

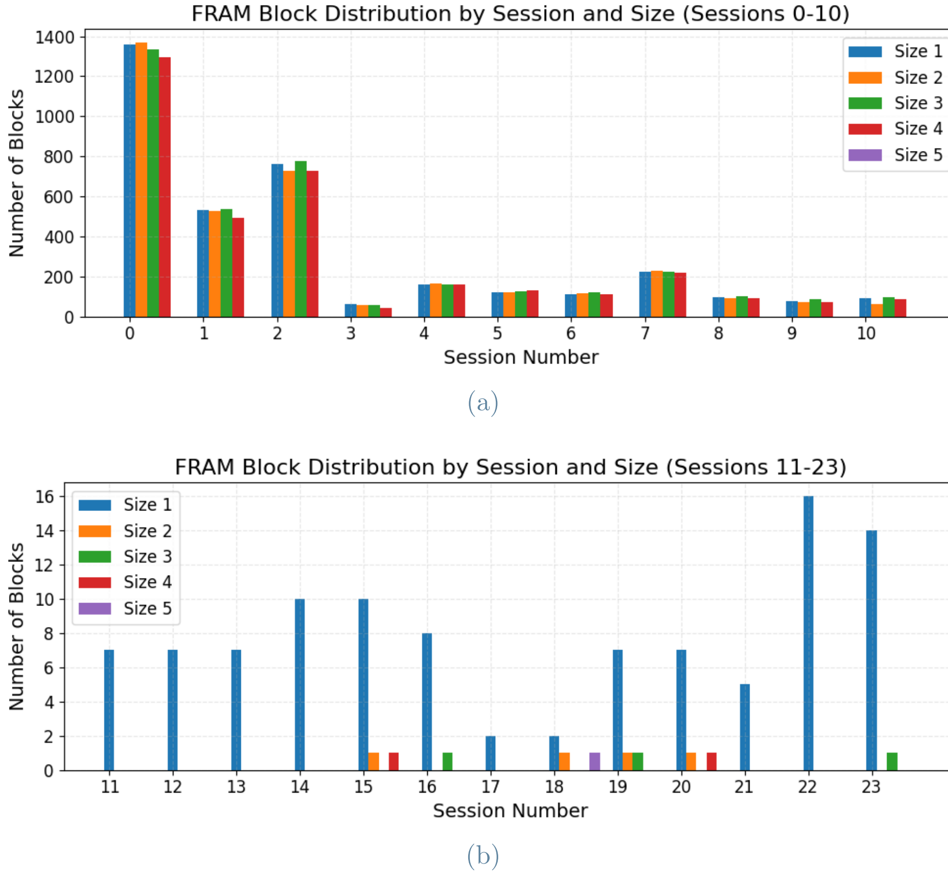


Figure 6.5: FRAM block size distribution over sessions.

and Figure 6.5b illustrate the trend of block sizes across the sessions, where the x-axis represents the sessions and the y-axis shows the number of blocks of each size detected during each session. Different colors correspond to different block sizes.

We observe that until the 11th activation session, the number of blocks of sizes 1 to 4 remains consistent across sessions. After this point, the trend changes, with a decrease in faults and a predominance of blocks of size 1. Comparing this trend with those observed in Figure 6.2a, Table 6.2, and with the observed regular block distribution, some similarities become evident. FRAM memory exhibits different fault dynamics during the first 11 sessions, before stabilizing.

A different distribution is observed for MRAM, where in some sessions the number of blocks of size 2 slightly exceeds those of size 1, as illustrated in Figure 6.6a. However, these instances are rare and the differences are minimal, so we cannot confirm the presence of a distinct trend. Even in this case, blocks of size 1 remain the most frequent.

A similar pattern is observed for ReRAM. As presented in Figure 6.6b, the distribution consistently shows a predominance of blocks of size 1, with larger blocks appearing less

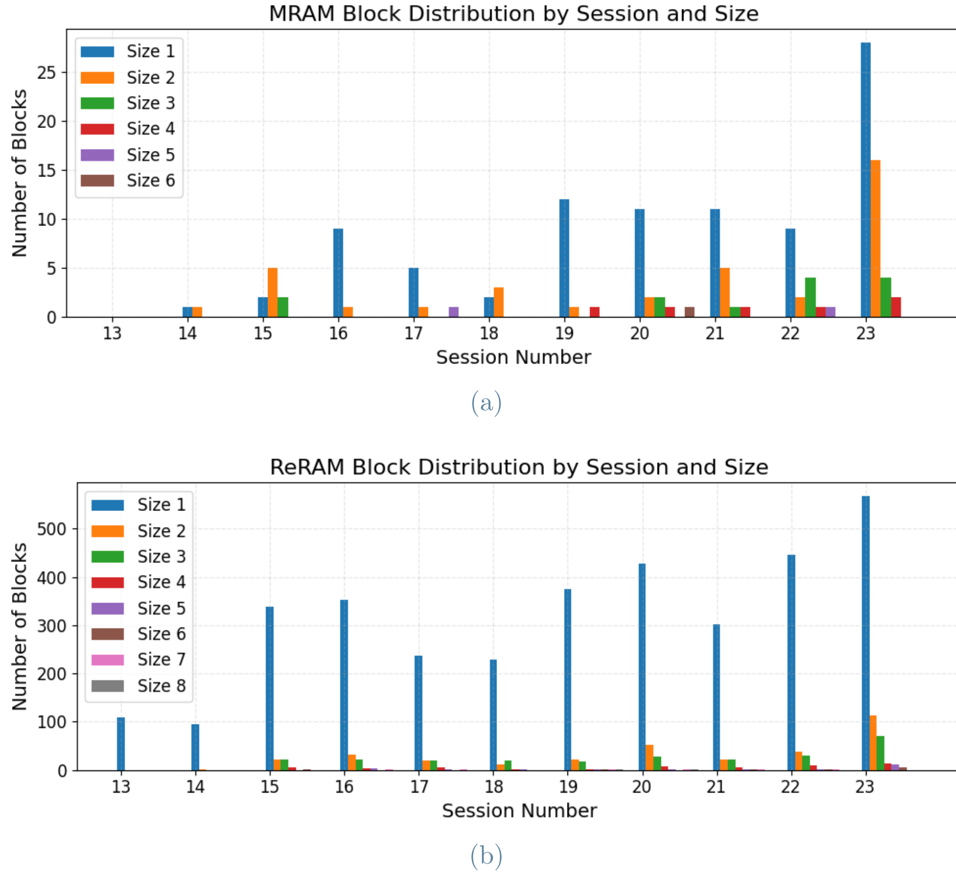


Figure 6.6: MRAM and ReRAM block size distribution over activation sessions.

frequently as their size increases.

For a clearer view of the overall block size distribution, Figure 6.7a presents the FRAM block sizes observed before the 11th activation session, while Figure 6.7b shows the distribution from the 12th session onward. In the first case, blocks of sizes 1, 2, 3, and 4 occur with roughly the same probability. In contrast, the second case reveals a different pattern. Blocks of size 1 become significantly more frequent, and the probability of occurrence progressively decreases as the block size increases.

A similar trend is observed for the MRAM and ReRAM block size distributions, shown in Figure 6.7c and Figure 6.7d, respectively. In both cases, blocks of size 1 are the most frequent, with the probability gradually decreasing as the block size increases.

Combining these observations with those from the previous sections, we can clearly distinguish between the first and second halves of the nanosatellite mission. During the first 11 activation sessions, only FRAM data is available, and it exhibited unique characteristics in terms of fault rate, spatial distribution, and block size distribution. After this period, its fault dynamics begin to resemble those of the other two memories. From that point

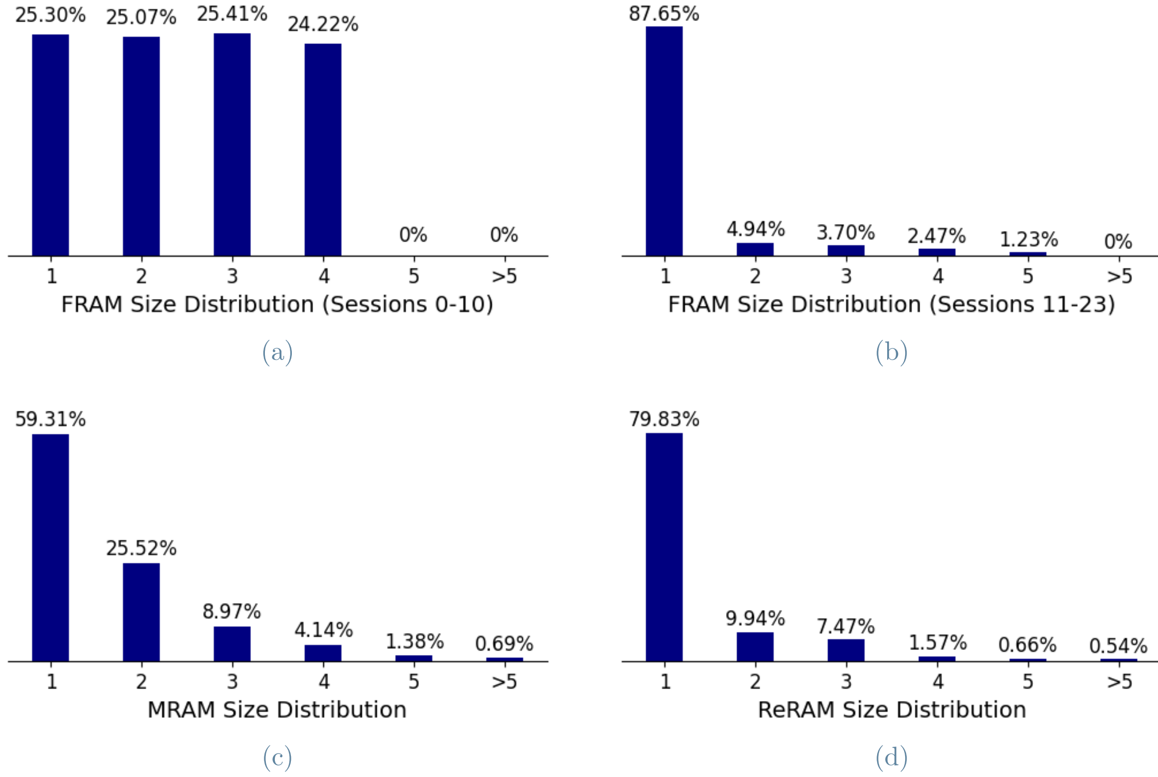


Figure 6.7: Block size distribution across memories.

onward, data from MRAM and ReRAM also becomes available, and FRAM's behavior aligns with theirs, showing a more uniform distribution in both fault frequency and block sizes.

Duration. Each block is also associated with a duration, as it can persist across multiple consecutive memory dumps. We observe that a block may appear multiple times throughout the mission, each time with a potentially different duration. By analyzing the distribution of durations across all memory dumps in sequence, according to the order in which they were received from the satellite, we can compute the duration distributions.

For the FRAM memory, we divided the ordered sequence memory dumps into two parts, as we did when calculating the size distribution, due to the observed differences in behavior between the first and second phases of the mission.

Figure 6.8a shows the block duration distribution for the first 168 memory dumps, while Figure 6.8b illustrates the distribution for the remaining dumps. The distributions clearly differ between the two phases. In the first phase, almost all blocks have a duration of 1, whereas in the second phase, blocks tend to have longer durations, indicating that faults persist in memory for a longer period. This behavior further supports the presence of two

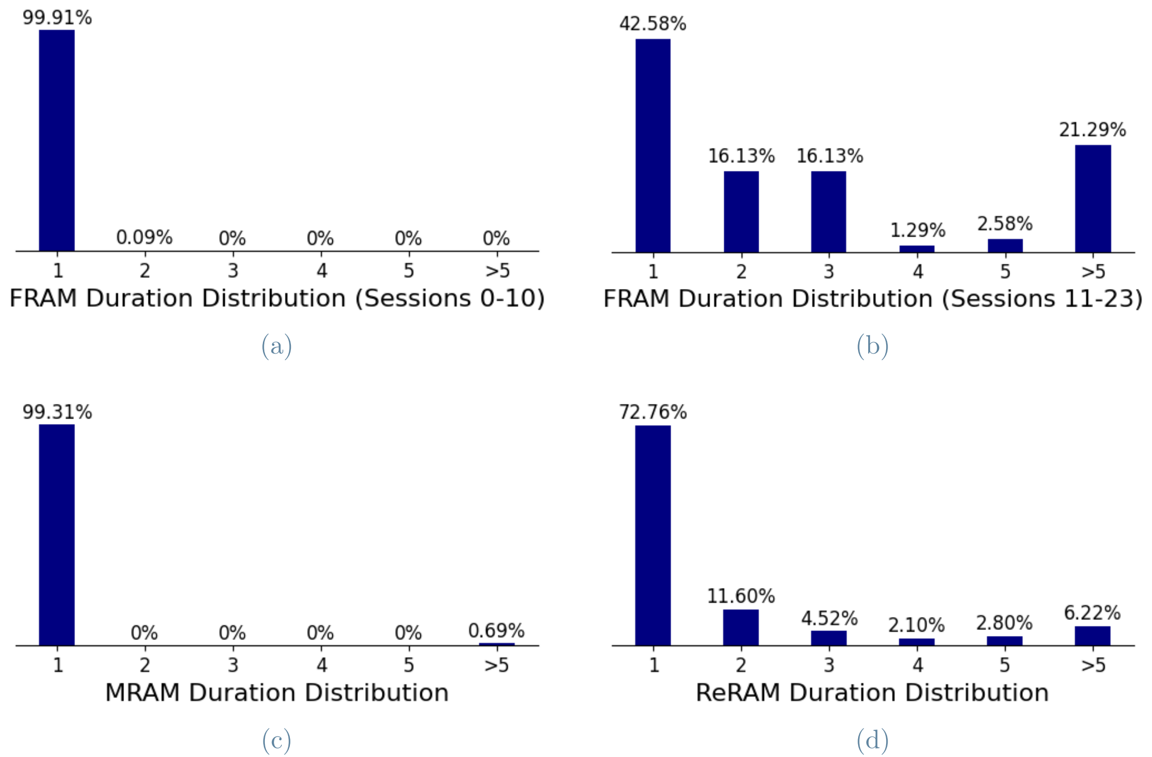


Figure 6.8: Block duration distribution across memories.

distinct fault dynamics in the FRAM memory.

Figure 6.8c illustrates the block duration distribution for MRAM. Most blocks have a duration of 1, with only a few cases showing longer durations, indicating that nearly all faults disappear very quickly.

Finally, Figure 6.8d shows the distribution for ReRAM. In this case, blocks persist across more memory dumps, and the probability of observing durations greater than 1 is higher, decreasing as the duration increases.

Classification. Transient faults can be classified as either SEUs or SEFIs, as discussed in Section 2.2. The classification approach involves analyzing the last memory dump of each session to determine whether a block disappears after the reset, that is, after the nanosatellite powers off, as explained in Section 5.2.

We check whether a block that appears in the final dump of a session persists in the first dump of the following session. If the block disappears after the nanosatellite has been powered off, we classify it as a SEFI. Instead, all the other blocks that persist when the nanosatellite powers off or vanish within the same activation session are classified as SEUs.

We observed that a large number of blocks persist for only one or two consecutive memory dumps. These often appear near the end of an activation session and then disappear. However, similar short-lived blocks are also found in other parts of the activation sessions, exhibiting the same brief duration and disappearance pattern. Because of this, combined with the fact that we cannot know the exact time between the end of an activation session and the start of the next one, it becomes impossible to reliably distinguish between SEU and SEFI. There is no consistent behavior that allows us to determine whether a fault disappeared due to the nanosatellite powering off, as would be expected for a SEFI, or if it naturally vanished without any external intervention, as would be expected for a SEU.

6.3.5. Heatmaps

After analyzing the block features and classifying them into transient or permanent, we create heatmaps that highlight the memory regions most vulnerable to corruption. Heatmaps are matrices representing the memory array, where each cell indicates the probability that the corresponding bit will experience a fault, meaning the likelihood of a bit flip occurring at that location.

Darker colors are assigned to bits with higher probabilities of corruption. A color bar next to each heatmap indicates the probability values corresponding to each color shade. In addition, previously observed permanent bit flips are marked in black to visually indicate the specific memory bits that have permanent damage.

Figure 6.9 highlights the most corrupted regions of the FRAM. We observe that corruption is present in many areas of the memory, but generally with very low probability and without any specific regions appearing more vulnerable than others. However, there are small regions marked with darker colors, indicating a higher likelihood of corruption. In these cases, the darker areas correspond to blocks with a duration longer than a single memory dump, meaning they are repeated across multiple consecutive dumps.

Figure 6.10 shows the heatmap of a portion of the MRAM memory, highlighting both the single permanent fault and the distribution of the most corrupted memory regions. A distinctive behavior observed in MRAM is that bit flips in individual memory dumps appear sparse. However, when considering the overall trend, the most corrupted regions are not randomly scattered across the memory. Instead, they tend to cluster in nearby areas, suggesting a degree of spatial locality in the occurrence of faults.

For the last heatmap, a portion of which is shown in Figure 6.11, related to the ReRAM, we observe a different spatial distribution. Compared to the other two memory types, ReRAM has a higher total number of bits, so the faults appear less dense visually. Also

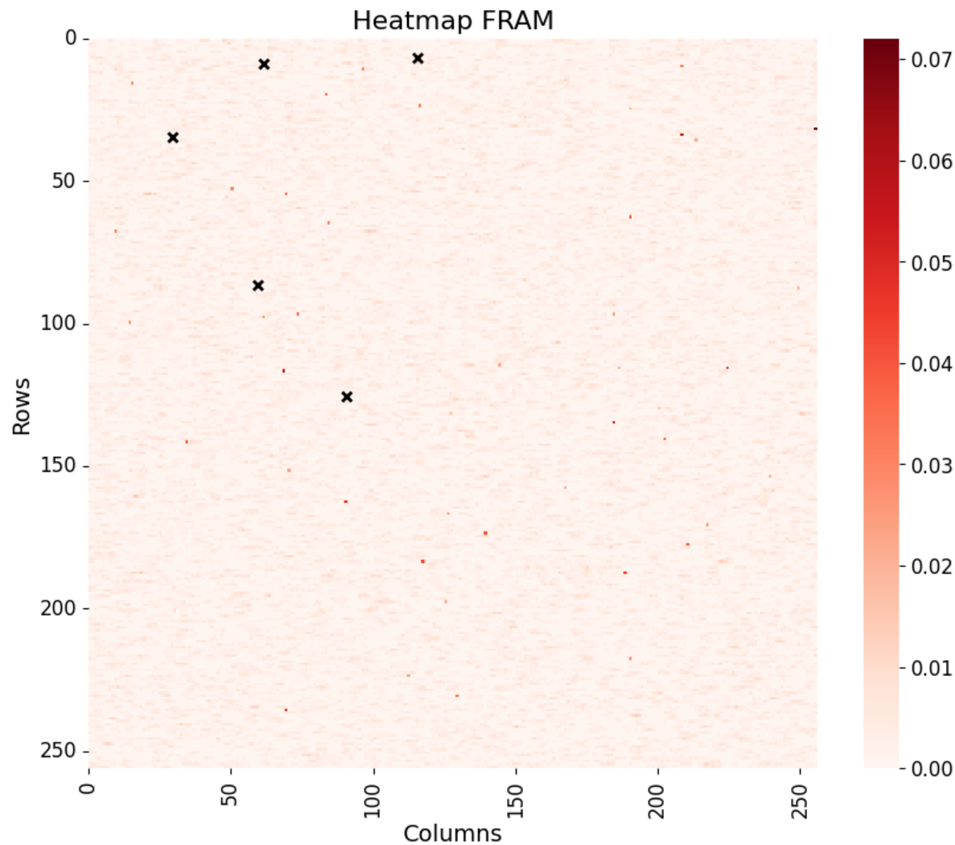


Figure 6.9: FRAM corruption heatmap.

in this case, no specific regions appear more corrupted than others; instead, the faults seem to be randomly distributed.

6.4. Memory Comparison

To conclude the analysis, we present a comparison across all memories, summarizing their respective advantages and disadvantages based on the results and observations discussed above.

As shown in Table 6.2, FRAM has the lowest storage capacity, followed by MRAM and then ReRAM. Considering the absolute values of the mean and standard deviation, FRAM and MRAM exhibit similarly stable behavior, with a very low number of bit flips and fluctuations around the mean. ReRAM, on the other hand, shows much higher absolute values compared to the other two memories, along with more pronounced fluctuations.

Looking at the same Table but expressing these values as percentages, MRAM shows the lowest mean percentage. This is because, in absolute terms, it has values similar to

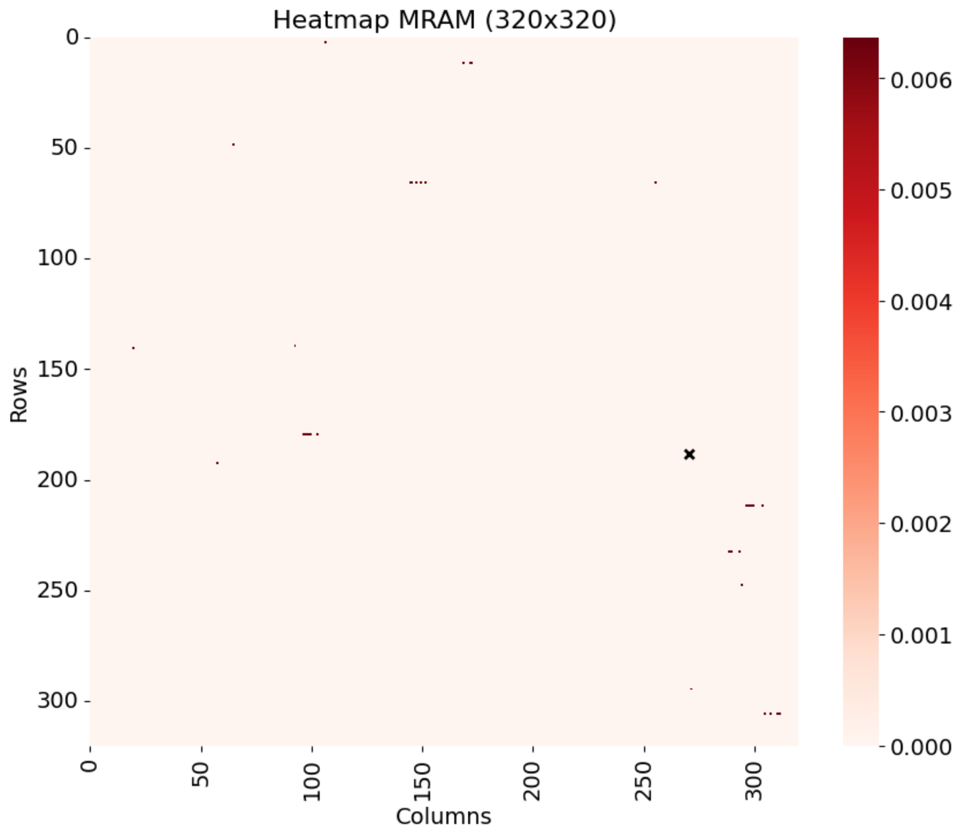


Figure 6.10: MRAM partial corruption heatmap.

FRAM, but when related to the total number of bits stored, the corruption percentage is lower due to MRAM's larger storage capacity. The same reasoning applies to ReRAM: although it has the highest number of faults in absolute terms, its percentage of corrupted bits is lower than that of FRAM, thanks to its significantly larger storage capacity.

In all memories, bit flips are organized into horizontal blocks, apparently independent from each other and varying in both size and duration. A subset of these blocks has been classified as permanent. The highest number of such faults was observed in FRAM, with five permanent bit flips, followed by ReRAM with three, and MRAM with only one. In most cases, across all three memories, these faults are of the stuck-at-1 type, indicating a higher vulnerability for this logical state.

No such prevalence of one logical state over the other is observed for blocks classified as transient, which occur on both bits initially set to 0 and to 1. These transient blocks show similar size distributions across the three memories. Blocks of size 1 are the most frequent, and the probability of occurrence decreases as block size increases. Regarding block durations, FRAM and ReRAM exhibit similar dynamics, with a prevalence of blocks of duration 1, but also a presence of blocks that persist for longer periods. In contrast,

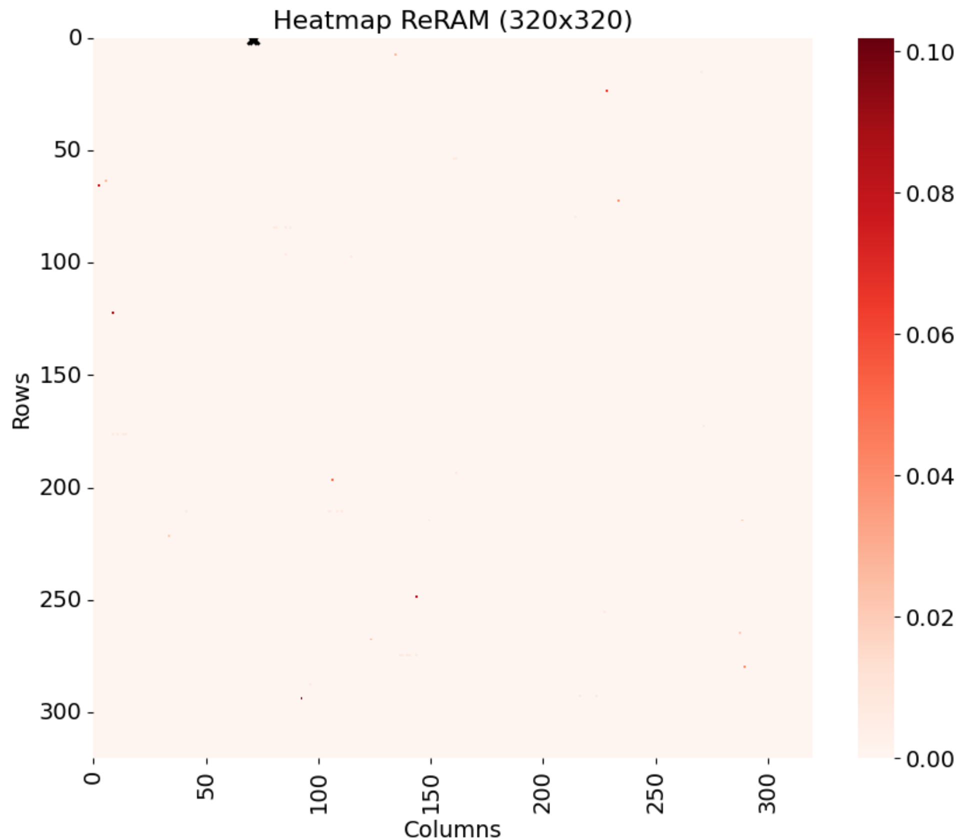


Figure 6.11: ReRAM partial corruption heatmap.

MRAM shows a different behavior, with almost all blocks persisting for only a single memory dump and quickly disappearing.

In conclusion, MRAM appears to offer the best compromise in terms of corruption levels and storage capacity, although it requires more power compared to the other two memories, as shown in Table 4.1. Its faults tend to disappear quickly, and only a single permanent bit flip was observed.

ReRAM, on the other hand, is suitable for applications requiring high storage capacity. However, its transient faults can persist for long periods, and the total number of bit flips is higher compared to the other memories. Additionally, its power consumption is lower compared to the other two memories, as indicated in Table 4.1.

Finally, FRAM has the smallest storage capacity and the highest corruption percentage, although its absolute fault counts are similar to MRAM. It exhibits the largest number of permanent bit flips and can reach very high corruption levels if issues occur in its memory controller. Nevertheless, it remains one of the most energy-efficient options, as shown in Table 4.1.

7 | Fault Injection

In this chapter, we describe the techniques used to develop fault models and generate faults with characteristics similar to those observed in real data. These methods can be applied in simulations to study the behavior and resilience of systems during potential ground-based experiments.

Section 7.1 describes the methods derived from the statistical observations of the real data. In Section 7.2, we then discuss another approach based on machine learning and data generation using a deep learning model, specifically a generative neural network. Finally, Section 7.3 presents the design of the interface developed to utilize the generated fault models.

The following techniques focus exclusively on simulating transient faults, since we observed in Section 6.3 that permanent faults appear to occur randomly, typically affecting at most one or two bits. As a result, it is not possible to accurately model their occurrence.

7.1. Statistical Approach

The goal is to develop a fault model that can be used to reproduce the distribution of faults observed in the real experiment onboard the nanosatellite.

This specific fault model uses a statistical technique, starting from the observation of the distribution of faults described in Chapter 6. This technique is based on the observation that bit flips tend to occur in blocks, and each block appears to be independent of the others, as discussed in Section 6.3. For these reasons, we can treat blocks as independent from each other and consider each of them as a single entity. Block's features, such as its position in the memory array, size, duration, number of occurrences, and probability of appearance, must be taken into account to introduce faults that reflect the characteristics of those observed in the real data.

We decided to create two different statistical fault models based on real observations. The first model does not take sequential dependencies into account and can be used in contexts where faults are introduced only once and timing is not important. The second

model includes the duration of each block and is suitable for simulation contexts where the model is applied consecutively, making it necessary to reflect the real observed sequential behavior.

7.1.1. Static Fault Generation

We developed a first static fault model by analyzing each observed block, focusing on its frequency and the probability of its occurrence.

First, a sequence of 0s is initialized with the same length as the real memory sequence. This sequence represents a clean bit flip mask, with no faults present. In the next step, faults are introduced by converting selected 0s to 1s based on the following statistical method. For each observed transient block, a random number between 0 and 1 is generated. If this number is lower than the block's associated occurrence probability, the block is inserted into the simulated mask, placing 1s in the cells affected by this fault.

The goal of this method is to preserve the spatial characteristics, size, and location of the faults, while reproducing their occurrence frequencies according to the observed probabilities, enabling a realistic simulation of transient fault behavior.

Cells that were never observed to flip are left untouched, as we cannot determine whether their lack of corruption was due to the intrinsic reliability of the hardware or simply the limited duration of the experiment.

7.1.2. Sequential Fault Generation

We also developed a sequential fault model, similar to the static model, that relies on the occurrence probability of each block. However, it extends this approach by incorporating the duration of each block, that is how long a fault persists across consecutive memory dumps. This allows the model to not only simulate the likelihood of a block occurring but also to reproduce its sequential behavior more accurately, reflecting how long the fault remains active once it appears.

Each observed block has one or more durations based on the number of occurrences, and this duration can vary each time the block appears. Since the available data do not allow us to determine whether other durations might occur, each block is assigned a duration randomly selected from the observed ones, as there is insufficient information to establish whether certain durations are more likely than others.

Each time a block is inserted into the simulated bit-flip mask, one of its observed durations

is chosen randomly, and this value determines how many consecutive times the block will persist in the subsequent generated masks.

When generating a new bit-flip mask, the initialization step accounts for blocks from the previous mask that are still within their assigned duration. These blocks are retained in the simulated mask before introducing new ones using the previously described statistical method, preserving their duration for subsequent mask generation.

7.2. Machine Learning

We developed an additional fault model using a deep learning approach, specifically a VAE whose structure and functioning are described in Section 2.5.

This approach is only feasible using data referred to the FRAM memory. As discussed in Section 6.3, blocks in FRAM exhibit a specific pattern distribution, and a larger amount of data is available for the neural network to learn from compared to the other two memories. Additionally, bit flips in the other memories are very sparse relative to their size. The available data is roughly half that of FRAM, and the blocks do not appear to follow a consistent pattern that the network could learn from.

The VAE is created using Tensorflow [18] and it is composed of two connected neural networks, called *encoder* and *decoder*. The encoder takes an input and compresses it into an intermediate representation called the *latent space*. The decoder then starts from this compressed representation to reconstruct the original input. After the training phase, we can use the decoder alone: by sampling random elements from the latent space, we can generate new data that shares the same features and distribution as the data the VAE is trained on.

The choice of this specific network comes from the need to address our main challenges, such as the small dataset available for training, the impossibility of applying data augmentation techniques, and the sparsity of bit flips. In fact, each bit flip mask used for training contains a very low number of 1s compared to 0s, reflecting the rarity of faults. Standard data augmentation techniques commonly used for images, such as shifts, rotations, or zoom, are not applicable in this context. This is because we cannot determine whether certain memory cells never experienced bit flips due to the inherent robustness of the hardware or simply because the experiment duration was not long enough to expose them. Therefore, repositioning bit flips from their observed locations in the training set could introduce unrealistic patterns, placing faults in memory regions that may never exhibit failures in practice.

Given the very limited amount of data available, our objective is not to build a model that generalizes well to completely new data, but rather to accurately learn and reproduce the specific characteristics of the dataset we have. Deep learning models typically require tens or hundreds of thousands of examples to generalize effectively, which is unfeasible in our context due to the difficulty of collecting nanosatellite data. Consequently, we prioritize precise learning of the existing data, accepting limited generalization in exchange for a faithful representation of the observed phenomena.

7.2.1. Neural Network Design

The first step before the training and generation phases is designing the neural network. Due to the small dataset, we must select a network architecture with a low number of parameters to avoid overfitting. Additionally, the loss function must address the issue of sparsity, as the data contains far fewer ones than zeros.

The input data is treated like images by reshaping all the bit-flip masks into a square format. This allows us to leverage existing techniques commonly used in pattern recognition and image generation tasks. The following list outlines all the implementation choices made to achieve our objectives.

Layers type. We design the VAE using both *convolutional and dense layers* in the encoder and decoder, creating a symmetric structure for these two parts of the network.

Convolutional layers are well-suited for learning local patterns and have been extensively applied for tasks involving image-like data, which aligns with our representation of bit-flip masks as 2D matrices [53]. Another reason for using convolutional layers instead of dense layers is that dense layers significantly increase the number of trainable parameters, which, given our limited dataset, carries a high risk of overfitting. Moreover, our objective is to focus on the spatial locality of faults, capturing potential dependencies between those occurring in nearby memory regions.

The encoder consists of three convolutional layers [3], each with progressively increasing filter sizes. Following the final convolutional layer, the output is flattened and passed through two dense layers [6]: one that computes the mean, and another that calculates the log variance of the latent variables. These two outputs are then fed into a custom layer [13] implementing the reparametrization trick, which samples the latent vector from a normal distribution parameterized by the learned mean and variance.

The decoder receives this sampled latent vector and first maps it back to the appropriate shape via a dense layer, followed by a reshape operation. It then passes through three

convolutional transpose layers [4] with decreasing filter sizes, mirroring the encoder's structure in reverse. This design enables the network to reconstruct the original input shape through learned upsampling.

Activation function. The model employs two different activation functions at different stages of the network to meet specific functional requirements, Rectified Linear Unit (ReLU) and Sigmoid [44].

All the intermediate layers, both in the encoder and decoder, use the ReLU activation function. ReLU is chosen for its simplicity and efficiency. It speeds up convergence, reduces the risk of vanishing gradients, and provides non-linearity while remaining computationally inexpensive. This makes it well-suited for compact networks that work with limited data and sparse inputs. For the output layer of the decoder, we use the Sigmoid activation function because its output is bounded between 0 and 1, making it a suitable choice for representing the output probabilities of bit flips. It ensures that each cell in the output mask represents the likelihood of a bit flip occurring at that memory cell, allowing it to be converted into a binary format through thresholding.

Loss function. The loss function used during training combines the *weighted binary cross-entropy (WBCE) loss* with the *Kullback-Leibler (KL) divergence*.

The WBCE is essential for handling data sparsity, where bit-flip masks have many more zeros than ones. Without it, standard binary cross-entropy would bias the model toward predicting zeros, neglecting the rare but important ones that indicate actual bit flips. To mitigate this, a positive weight is applied to the 1s during training. This ensures that the model learns to pay special attention to these rare events and helps it reproduce their statistical occurrence more accurately. The formula of the WBCE is the following [70]:

$$\mathcal{L}_{\text{WBCE}} = -\sum_{i=1}^{N} \left[w_1 \cdot y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i) \right] \tag{7.1}$$

where:

- $y_i \in \{0, 1\}$ is the true label for the i-th bit,
- $\hat{y}_i \in (0, 1)$ is the predicted probability of a bit flip,
- $w_1 > 0$ is a weight assigned to the positive class to address class imbalance,
- N is the total number of prediction targets.

The KL divergence term, instead, plays a crucial role in regularizing the latent space. It

forces the distribution of the latent variables to approximate a standard normal distribution, encouraging smoothness and continuity in the latent space. This regularization is key to enabling the decoder to generate realistic and coherent outputs even from randomly sampled latent vectors [42]. The KL divergence term is given by the following formula [42]:

$$\mathcal{L}_{\text{KL}} = -\frac{1}{2}\sum_{j=1}^{d}\left(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2\right) \tag{7.2}$$

where:

- μ_j and σ_j^2 are the mean and variance of the latent variable z_j,
- d is the dimensionality of the latent space.

The total loss minimized by the model combines the 7.1 and 7.2, weighted by an hyper-parameter λ:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{WBCE}} + \lambda \cdot \mathcal{L}_{\text{KL}} \tag{7.3}$$

In this work, we set λ to 1, giving equal importance to both reconstruction accuracy and latent space regularization.

7.2.2. Data Generation

After defining the design of the VAE architecture, we trained the network by dividing the entire dataset into 90% for the *training set* and 10% for the *validation set*. We do not further split the dataset to create a *test set*, due to its limited size. In fact, doing so would significantly reduce the amount of data available for training, making the training phase unfeasible.

The Figure 7.1 illustrates the steps required to obtain a network capable of generating new synthetic data. After the training phase, the network is able to reconstruct data. Therefore, if a binary matrix is given as input, it can reconstruct it using the encoder's compression, the subsequent mapping in the latent space, and the decompression performed by the decoder.

By using only the decoder, we can generate new data similar to that used during the training phase. The idea is to select a random vector for sampling in the latent space. As previously explained, the specific loss function of the VAE allows the latent space to remain

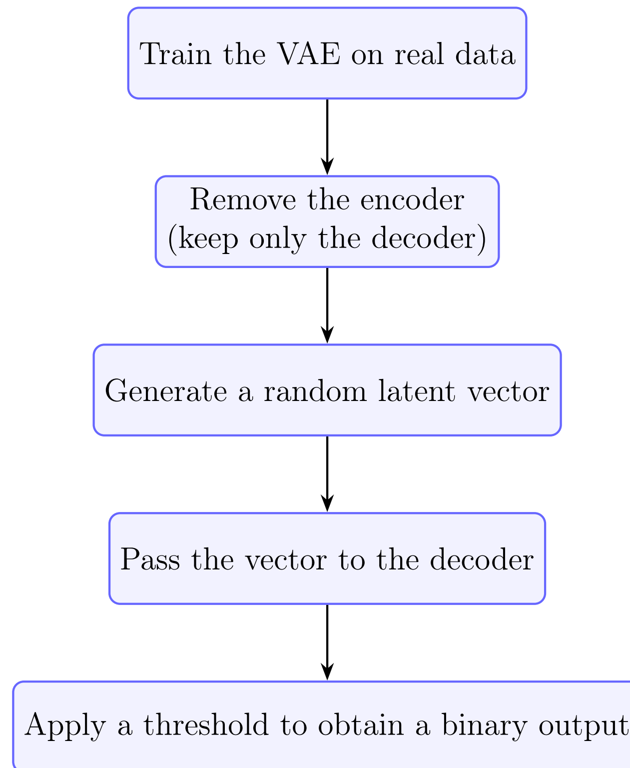


Figure 7.1: Fault generation workflow using the VAE.

continuous and similar to a normal distribution. By sampling from this distribution, we can generate new data resembling that used in training.

A random vector of the same size as the latent space is provided as input to the decoder, which then generates new data. The output is a matrix with the same dimensions of the input, and each cell contains a probability representing the likelihood that the corresponding bit is flipped. By applying an appropriate threshold to this probability, we can obtain a binary matrix that represents a bit-flip mask. The procedure and the results for the threshold choice are discussed next, in Section 8.3.

7.3. Interface Design

To allow the use of these models, we create an interface that, in our specific case, consists of a single operation. A binary sequence representing the logical state of the memory is provided as input, and faults are introduced according to the selected fault model. The function returns the modified sequence, with bits flipped at the positions specified by the chosen fault model.

To perform this task, the function takes the following inputs:

- **Binary string**: the logical representation of the memory where the user intends to introduce faults.
- **Memory type**: this parameter can take only three string values, *FRAM, MRAM*, or *ReRAM*, and is used to select the appropriate memory model.
- **Injection type**: this parameter accepts one of three string values, *STATIC, SEQUENTIAL*, or *ML*, to choose the fault model described in the previous sections.
- **Mask Entries**: the faults to be inserted during the initialization phase of the bit flip mask. If the user selects the sequential fault model, this field includes the faults carried over from the previously generated bit flip mask; otherwise, the value is empty.

The output of this operation is the same binary string provided as input, but with specific bits flipped according to the selected fault model. The internal logic that determines which bits are flipped is hidden from the user, who only sees the final result: a binary string with faults introduced at particular positions. Additionally, the operation returns the list of blocks inserted during bit flip mask generation, along with their simulated durations. If the selected method does not account for duration, this additional return value is empty.

8 | Evaluation

In this chapter, we introduce the metrics used to evaluate the fault models presented in Chapter 7.

Section 8.1 defines these metrics and presents the statistical properties that the models must preserve. Section 8.2 describes the methods used to compare the metrics defined in the previous section with those derived from the real data. Section 8.3 discusses the parameter selection process for the machine learning approach. Finally, Section 8.4 presents and analyzes the evaluation results based on the previously defined metrics.

8.1. Metrics Definition

We evaluate the fault models using fundamental statistical metrics that describe fault properties, generating 200 bit-flip masks for each model. This number provides a balance between statistical reliability and comparability with the available datasets, which include 376 valid memory dumps for FRAM and 158 each for MRAM and ReRAM, as shown in Table 6.1.

When evaluating the statistical models for FRAM, we do not consider all the real bit-flip masks, but only those corresponding to the activation sessions for which MRAM and ReRAM memory dumps are also available. This ensures a direct and consistent comparison across memory technologies under the same operating conditions.
For the machine learning model, created specifically for the FRAM memory as discussed in Section 7.2, we instead use all available data. This allows the VAE to be trained on the largest possible dataset and enables the network to learn all potential spatial patterns of faults.

The following metrics are calculated for all the sequences of generated bit-flip masks and then compared with those observed in the real data:

- Mean: the average number of flipped bits (1s) per mask. The same formula used for the real data is applied here (Equation 5.2).

- STANDARD DEVIATION: this metric quantifies the variability of the bit-flip count around the mean across the generated masks. Again, the same formula used for the real data is applied (Equation 5.3).
- BLOCK SIZE DISTRIBUTION: this metric describes how block sizes are distributed. As defined in Section 5.1, each block has an associated size. Considering all generated bit-flip masks, we count the number of blocks of each size and compute their percentage distribution.
- BLOCK DURATION DISTRIBUTION: this metric describes how long blocks persist across consecutive masks. For each block, we record its duration, defined as the number of consecutive masks in which it appears. Considering all generated masks, we then compute the percentage distribution of block durations.

The mean and standard deviation verify that both the overall severity and variability of bit flips are accurately captured. Comparing these metrics across FRAM, MRAM, and ReRAM further confirms that the model generalizes to different memory technologies, providing a reliable basis for evaluating system robustness without under- or overestimating fault impacts.

It is also important to verify that the models accurately reproduce both the block size and duration because faults in memory are often correlated. Correctly modeling these properties ensures that the fault model captures realistic patterns of consecutive faults, which directly impact the effectiveness of error-correction mechanisms and the overall reliability of the system. Failing to reproduce block size or duration could lead to misleading conclusions about system robustness.

8.2. Statistical Comparison

Once the masks are generated, we evaluate if the models preserve the general statistical properties described in Section 5.1. The values obtained are then compared with the corresponding real measurements reported in Chapter 6 using the following approaches:

Mean and Standard Deviation: we compute the average number of bit flips and the corresponding standard deviation for the generated masks. These values are then compared numerically with the metrics obtained from the real data. Small differences indicate that the model preserves both the average behavior and variability of bit flips observed in the real system.

Block size and duration distributions: To quantitatively compare the block distributions generated by the fault models with the real memory data, we use the *Wasserstein distance* [17, 56]. This metric is particularly appropriate for our case because it focuses on probability mass functions, and it works even if the distribution may involve small counts per class.

This distance measures the minimal effort required to reconfigure the probability mass of one distribution to recover the other distribution [56]. In our context, it quantifies the difference between the distributions of block properties (e.g., block size or duration) in the real and generated datasets. A smaller distance indicates that the generated distribution closely approximates the real distribution, whereas a larger distance highlights discrepancies.

To apply this metric, we first compute the empirical distributions of block counts across classes for both real and generated data. We then calculate the distance between these distributions for each block property.

A limitation of the raw Wasserstein distance is that its magnitude depends on the scale of the underlying variable. For example, a distance of 2 is relatively large if the variable ranges from 1 to 6, but negligible if it ranges from 1 to 1000.

To enable comparisons that are independent of scale, the Wasserstein distance can be *normalized* by the maximum possible distance between two distributions defined on the same support. Formally, the normalized distance is given by:

$$W_{\text{norm}} = \frac{W}{W_{\text{max}}}$$

where W is the Wasserstein distance between the two distributions and

$$W_{\text{max}} = x_{\text{max}} - x_{\text{min}}$$

is the maximum achievable distance, obtained when one distribution assigns all probability mass to the minimum support value x_{min}, while the other assigns all mass to the maximum support value x_{max}. The results can be interpreted as follows:

- $W_{\text{norm}} = 0$: the distributions are identical.
- $0 < W_{\text{norm}} \ll 1$: the distributions are very similar; only a small fraction of the maximum possible probability shift is required.
- $W_{\text{norm}} \approx 1$: the distributions are maximally different, with their probability mass located at opposite extremes of the support.

For instance, a value of $W_{\text{norm}} = 0.03$ indicates that the two distributions differ by only 3% of the maximum possible displacement of probability mass, and can therefore be considered nearly identical in practical terms.

This definition and normalization make the Wasserstein distance a robust and interpretable metric for assessing similarity between probability distributions, particularly in discrete settings. All the following results refer to the W_{norm}, allowing for a more immediate and intuitive evaluation.

8.3. Parameters Selection

This section presents the selection of parameters for the VAE used in the machine learning–based fault model.

We selected the hyperparameters of the neural network, including the number of layers, the number of neurons per layer, the learning rate, and the weight for the weighted binary cross-entropy, through an empirical tuning process. We iteratively tested different configurations and evaluated them on validation data, and we chose the combination that provided satisfactory performance for the experiments.

The main challenge, however, was the selection of the threshold for the probability matrices output by the VAE, which are used to generate the bit-flip masks. The process of generating these masks is described in Section 7.2.

When using the Sigmoid function in the output layer, the default choice is to apply a threshold of 0.5 to classify each bit as flipped or not. However, in our case, we observed that many of the predicted probabilities are clustered around this threshold, making the classification ambiguous and potentially inaccurate. To improve this, we use the following data-driven strategy to choose the most suitable threshold. We define:

- **True Positives (TP):** bits actually flipped and correctly predicted as flipped.
- **False Positives (FP):** bits not flipped but incorrectly predicted as flipped.
- **False Negatives (FN):** Bits flipped but not detected by the model.
- **True Negatives (TN):** Bits not flipped and correctly predicted.

Then, to evaluate the quality of the reconstructed bit-flip masks, we use the F1-score, which is the harmonic mean of *precision* and *recall*, whose definitions in our context are:

- **Precision:** it measures the proportion of predicted bit flips that are actually correct:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{8.1}$$

- **Recall:** it measures the proportion of actual bit flips that are correctly identified:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{8.2}$$

- **F1-score:** it provides a balance between precision and recall:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8.3}$$

The goal is to find the threshold that maximizes the F1-score, which provides a balanced measure of the model's ability to correctly identify flipped bits while minimizing both false positives and false negatives. This metric is particularly useful when the classes are imbalanced, as it penalizes models that perform well on only one aspect (e.g., high recall but low precision). In our bit-flip detection task, maximizing the F1-score helps select a threshold that best balances the trade-off between detecting as many real flips as possible while minimizing false detections.

Due to the limited size of the dataset, we compute the F1-score using the entire dataset rather than reserving a portion for testing. As previously discussed, the data is extremely scarce, and further splitting would reduce the already small amount available for training.

In contexts where the available data is minimal, traditional train/validation/test splits may significantly reduce the number of meaningful observations, making reliable evaluation difficult. Our focus, therefore, is on accurately reproducing the observed data rather than generalizing to unseen samples.

Using this method, we found the optimal threshold to be 0.67, slightly higher than the conventional 0.5 threshold of the Sigmoid function. This suggests that increasing the threshold slightly helps include only those bit flips for which the network is more confident, while excluding ambiguous predictions.

To determine this value, we provided all the original bit flip masks as input, and the network reconstructed them. On these reconstructed matrices, we applied a fixed threshold, iteratively testing values from 0.5 to 0.8 in steps of 0.01. For each threshold, we computed the F1-score and, at the end of the process, we selected the threshold that achieved the

Mean

Memory	Real	Static	Sequential
FRAM	2,76	0,93	2,16
MRAM	1,52	2,18	2,36
ReRAM	95,27	97,07	91,46

Standard Deviation

Memory	Real	Static	Sequential
FRAM	2,53	1,20	1,42
MRAM	2,64	1,91	1,64
ReRAM	51,13	46,11	44,31

Table 8.1: Mean and standard deviation for each fault model across memories.

highest value. We chose 0.5 as the lower bound because it represents the standard Sigmoid threshold, and set 0.8 as the upper bound because higher thresholds would suggest that the network had not properly learned the masks.

8.4. Results

This section presents the results of the simulated statistical metrics, with a comparison across all memories, to evaluate each fault model for each memory, distinguishing between those generated by the statistical models and those produced by the machine learning model.

8.4.1. Statistical Models Results

Mean and standard deviation. We quantitatively compare the mean and standard deviation metrics with the real ones. Table 8.1 summarizes the results for each fault model and memory, which can be directly compared with those computed from the real data and presented in Table 6.2.

As the tables show, we observe that in all cases both the mean and standard deviation are similar to the real values, with absolute differences of one or two units for FRAM and MRAM, while ReRAM exhibits larger deviations. In ReRAM, the mean remains close to the original and the standard deviation is slightly lower, but when expressed as a percentage of the large storage capacity, it still remains very close to the real value.

We conclude that the statistical approach is accurate in preserving the basic metrics, as it models both the mean and standard deviation of the number of bit flips in a way that does not deviate significantly from the real values.

Block size and duration distribution. We compare the distributions of block size and duration with the real ones, and Table 8.2 presents all the computed normalized Wasserstein distances for each statistical model and memory. As the tables show, the static generation model accurately represents the size distributions but is less precise

Wasserstein distance for block size distribution

Memory	Static	Sequential
FRAM	0,006	0,003
MRAM	0,004	0,014
ReRAM	0,009	0,007

Wasserstein distance for block duration distribution

Memory	Static	Sequential
FRAM	0,237	0,015
MRAM	0,039	0,004
ReRAM	0,122	0,005

Table 8.2: Wasserstein distances for block size and block duration distributions for each fault model across memories.

in modeling the durations. This is because this method only introduces blocks without maintaining them across consecutive generated bit-flip masks. This occurs for the fault generations of all memories, but is less pronounced for MRAM because, as shown in Figure 6.8c, the majority of blocks have a duration of 1, so they disappear quickly without showing longer persistence.

In contrast, the modeling of duration distributions is more accurate in the sequential generation method for all memories, which also provides an accurate representation of the block size distributions for all three memories.

8.4.2. Machine Learning Results

We compute the same metrics after generating bit-flip masks using the machine learning model. The results for the mean and standard deviation are summarized in Table 8.3, this time compared with the mean and standard deviation of the real dataset presented in Table 6.2, considering all available data for the FRAM memory.

The model generates faults that roughly preserve the mean absolute number of bit flips, although it introduces, on average, fewer than ten additional bit flips. Regarding the standard deviation, this is not preserved, showing an absolute value roughly four times smaller than the real one.

By observing the generated bit-flip masks, we can observe that the network learns to generate blocks of different sizes and thus captures spatial correlation between bit flips.

The block size and duration distributions are also not accurately modeled, showing nor-

Mean

Real Mean	Generated Mean
96.05	102,62

Standard Deviation

Real StdDev	Generated StdDev
232.08	52,91

Table 8.3: Mean and standard deviation for machine learning model.

malized Wasserstein distances of 0.399 and 0.421, respectively. For the block size distribution, this is likely because the model can generate bit-flip masks containing blocks with different sizes, as in the real data, but it cannot learn how these sizes are distributed in relation to their frequency of occurrence.

Regarding the block duration distribution, the VAE by design does not account for the temporal relationship between one generation and the next, so the result is consistent with the nature of this deep learning model.

The main difference with respect to the statistical approach is that the network generates blocks even in regions where none were observed in reality. This is a positive aspect, as the network is producing faults that could potentially occur, based on the distribution of real faults mapped in the latent space, as explained in Section 2.5.

In our case, the problem is that, due to the inaccuracy of the model caused by the scarcity of training data, we cannot determine whether this behavior is because the network has not learned correctly, or if these are actual faults that could realistically occur in these positions derived from the study of observed fault locations.

9 | Suggestions for Future Missions

During our experiment, we encountered several issues, which led us to compile a list of suggestions for future missions. In this chapter, we introduce these suggestions to improve other possible experiments.

Section 9.1 explains the advantages of introducing an absolute concept of time. Then, Section 9.2 describes the need to collect more data. Finally, Section 9.3 highlights the importance of executing different algorithms onboard the nanosatellite.

9.1. Absolute Timestamps

In our experiment, we do not have an absolute concept of time, as discussed in Section 5.4. For the analysis, memory dumps are ordered by their arrival sequence, and the only available temporal information is the date and time marking the start of each transmission event from the nanosatellite.

This lack of precise timing limits our ability to detect time-based fault patterns, measure fault rates over specific intervals, and obtain accurate information about the duration of individual faults. Currently, fault duration is defined based on the number of consecutive memory dumps in which the fault persists, as discussed in Section 6.3. However, without knowing the actual time interval between dumps, this measure is only an approximation and may not reflect the true temporal span of a fault.

Moreover, the absence of absolute timestamps prevents the identification of temporal correlations between faults and external events, such as solar activity, orbital position, or environmental conditions. It is also impossible to detect periodic or recurring fault patterns and to compute accurate fault rates (e.g., faults per hour or day), which are essential for evaluating system reliability over time.

To overcome this limitation, future missions should consider implementing a method for distributing clock information to all boards or equipping each board with an independent but synchronized clock. This would allow each board to attach absolute timestamps to the transmitted data, enabling more accurate and meaningful fault analysis.

9.2. Dataset Size

The data we received from the nanosatellite is not sufficient to produce reliable statistical analyses or to train a neural network capable of generalizing well on the generated data. Therefore, the main recommendation is to collect more data.

In our case, as shown in Table 6.1, the number of valid observations is on the order of a few hundred, which limits the accuracy of frequency-based measurements and statistical evaluations. This limitation affects both the statistical confidence of frequency estimates and the performance of data-driven models, such as neural networks, which typically require large datasets to avoid overfitting and to generalize across unseen scenarios. In particular, rare patterns or long-term dependencies between faults may go undetected.

The main reason for the limited data availability is the complexity of the experimental setup. Both the satellite launch and the establishment of a reliable communication link with Earth are technically demanding and economically expensive operations. Moreover, the power generated by the solar panels must be shared among various onboard subsystems, which means that when the satellite is powered on, not all the energy is necessarily allocated to our experiment to initiate what we defined in Chapter 5 as an activation session.

Including multiple boards is necessary, as dedicating all available power to a single one would have been inefficient. Satellite missions involve high launch costs and offer limited opportunities for experimentation, making it essential to maximize the scientific results. Launching a satellite to conduct only one experiment would significantly reduce mission efficiency. By integrating multiple boards, the system runs several experiments in parallel, increasing the number of scientific results during the mission. This design choice, however, comes at the cost of reduced data availability for each experiment, as the available power and communication capacity must be shared among all active subsystems.

9.3. Onboard Algorithm Execution

In our experiment, the executed algorithm is always the same, which results in identical values and outputs being written to memory every time. Consequently, when bit flip masks are created, they are always based on the same reference binary sequence.

This approach restricts our ability to detect all the possible bit flips, because, as discussed in Section 5.4, some of them do not appear in the bit flip masks, potentially underestimating the true fault rate. As a result, some memory faults may remain hidden.

A possible approach to uncover hidden faults is to write different data patterns to memory by running different algorithms at each execution. With this method, it is still possible for a bit to experience a fault without being detected if the corrupted value coincidentally matches the expected value in the reference binary sequence. However, when a new algorithm is executed, the expected value at that specific memory location changes. As a result, the fault becomes detectable through the bit flip mask.

To address this, future experiments should incorporate different algorithms to reveal faults masked by consistent memory contents. Therefore, the current fault analysis may only reflect a subset of all possible bit flips, making it necessary to use more varied test conditions.

This situation presents a key trade-off: as the variety of system configurations and memory patterns increases, it becomes harder to collect a large amount of data for each specific case. This is because the total dataset gets divided across many different instances, which means fewer samples per case.

In conclusion, the current fault analysis captures only a subset of all possible bit flips, highlighting the need for a more varied testing approach. However, increasing the number of tests may reduce the dataset size for each case.

10 | Conclusion and Future Works

This thesis presents an experiment designed to collect data from three different non-volatile memories onboard a nanosatellite, specifically FRAM, MRAM, and ReRAM. The main challenges addressed in this work are the analysis of the collected data to study fault patterns and potential vulnerabilities of the memories, and the definition of fault models that faithfully represent the statistical distributions of observed faults, which can be used to perform fault injection operations that accurately reproduce the dynamics observed.

Analysis of the collected data shows faults in all three memories, both permanent and transient, typically characterized by bit flips grouped into blocks of varying size. Depending on their persistence, they can be classified as *transient blocks* or *permanent blocks*.

FRAM shows multiple phases in its bit-flips evolution, characterized by different means and standard deviations that gradually decline before stabilizing in a pattern similar to MRAM. ReRAM shows the highest absolute number of bit flips. Nevertheless, its large storage capacity keeps the percentage of corrupted data very low. MRAM, on the other hand, exhibits the most stable behavior and offers the best balance between the average number of bit flips and storage capacity. This makes it the most robust technology overall.

We observed that permanent blocks are uncommon and usually consist of a single bit, typically stuck at logical '1'. FRAM exhibited five such permanent bit flips, ReRAM three, and MRAM only one. This highlights that MRAM has the highest resistance to this type of fault.

For transient blocks, we noted that no particular logical state is more susceptible than the other, as flips occur with similar probability on both '0' and '1'. Smaller blocks are generally more frequent than longer ones, with the probability decreasing as block size increases. In terms of duration, MRAM blocks tend to be short-lived, often disappearing after a single memory dump. FRAM and ReRAM, in contrast, exhibit blocks that can persist across more than five consecutive memory dumps.

Overall, except for a few isolated permanent bit flips, no particular regions in any of the memories show increased vulnerability. The average corruption rate stays below 0.1%, confirming that all three non-volatile memory technologies exhibit high robustness in the space environment.

The fault models we used to reproduce fault distributions and their main characteristics are divided into statistical and machine learning models, both of which are capable of generating bit-flip masks. The statistical methods include one static model that ignores block durations and another sequential model that preserves them. In contrast, the machine learning model uses a VAE trained on bit-flip masks derived from the real memory dumps, to generate new masks that maintain the distribution of faults and their main characteristics.

Statistical methods aim to reproduce observed faults according to their statistical properties. Each fault is inserted into the bit-flip mask based on the frequency with which it was observed, ensuring that the model accurately reflects the distributions while preserving their size and duration as observed in the experimental data. For the VAE, the trained decoder is used to generate synthetic data, producing new bit-flip masks that contain faults with the same statistical properties as those observed onboard the nanosatellite in orbit.

We evaluate the models by generating 200 bit-flip masks for each technique, computing the mean, standard deviation, block size, and duration distributions, and comparing them with the real distributions using the Wasserstein distance.

The two statistical models preserve both the mean and the standard deviation for all memories. Both models also maintain the distribution of fault sizes, but only the sequential model can reproduce the distribution of fault persistence. The machine learning model can generate bit-flip masks with sequences of consecutive flips, consistent with real observations, and maintains a mean number of bit flips similar to the real data, with a difference of fewer than 10 bit flips compared to the real value. However, the standard deviations and the distributions related to fault size and persistence differ from those observed in the real data. This suggests that the network has learned spatial patterns but fails to accurately reproduce the other metrics except for the mean, likely due to the limited size of the training dataset.

From the results obtained, we can conclude that our fault models can reproduce faults while preserving their main statistical properties, each with a different level of accuracy. These values must be taken into account when selecting the method to introduce bit flips, considering the strengths and weaknesses of each technique.

Based on the results obtained, several directions for future work can be identified:

- Starting from the results of the analysis, it is possible to choose the most appropriate error correction techniques, optimizing them for the observed fault patterns and adapting them to the different types of non-volatile memories used in space. This is important because each memory technology exhibits distinct fault behaviors, including the frequency, distribution, and persistence of bit flips. Adjusting error correction mechanisms according to these characteristics can improve overall dependability.
- The developed fault injection interface, based on the created fault models, may be integrated into a more complex system to perform fault injection. This would allow, for example, testing the robustness of onboard software under realistic fault conditions, validating error-correction mechanisms, and evaluating system-level reliability without requiring access to the actual satellite hardware.
- This work supports the design of a nanosatellite for future missions, taking into account the suggestions derived from this experiment. These include the collection of a larger dataset to enable more accurate analysis and stronger statistical significance, which would also allow neural network–based approaches to improve generalization. Additional improvements could involve adding timestamps to memory dumps to enable temporal analyses or executing different algorithms to write diverse patterns into memory, potentially revealing faults that may not be detected with a single pattern.

Bibliography

[1] Endurosat official website. URL https://www.endurosat.com/.

[2] Satnogs - open satellite network. URL https://satnogs.org/.

[3] Conv2d layer documentation, . URL https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D.

[4] Conv2dtranspose layer documentation, . URL https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2DTranspose.

[5] What are smallsats and cubesats? URL https://www.nasa.gov/what-are-smallsats-and-cubesats/.

[6] Dense layer documentation. URL https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense.

[7] Cubesats, . URL https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/CubeSats.

[8] Types of orbits, . URL https://www.esa.int/Enabling_Support/Space_Transportation/Types_of_orbits.

[9] Radiation: Satellites' unseen enemy, . URL https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Radiation_satellites_unseen_enemy.

[10] Memoria nand flash. URL https://it.wikipedia.org/wiki/Memoria_NAND_flash.

[11] What are ferroelectric memories (feram)? URL https://techovedas.com/what-are-ferroelectric-memoriesferam/.

[12] Hubble space telescope mission. URL https://science.nasa.gov/mission/hubble/.

[13] Lambda layer documentation. URL https://www.tensorflow.org/api_docs/python/tf/keras/layers/Lambda.

[14] Mars cube one (marco) mission. URL https://science.nasa.gov/mission/marco/.

[15] Radiation: satellites' unseen enemy. URL https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Radiation_satellites_unseen_enemy.

[16] Endurosat 1u cubesat solar panel. URL https://satsearch.co/products/endurosat-1u-cubesat-solar-panel.

[17] Scipy wassertain distance. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wasserstein_distance.html. Accessed: 2025-09-20.

[18] Tensorflow official website. URL https://www.tensorflow.org/.

[19] James webb space telescope mission. URL https://science.nasa.gov/mission/webb/.

[20] D. Bank, N. Koenigstein, and R. Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, 2023.

[21] R. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 2005.

[22] K. Berahmand, F. Daneshfar, E. S. Salehi, et al. Autoencoders and their applications in machine learning: a survey. *Artificial Intelligence Review*, 2024.

[23] Bhagyashree, V. Kushwaha, and G. C. Nandi. Study of prevention of mode collapse in generative adversarial network (gan). In *IEEE 4th Conference on Information & Communication Technology (CICT)*, 2020.

[24] J. Bi, B. Li, et al. Total ionization dose and single event effects of a commercial stand-alone 4 mb resistive random access memory (reram). *Microelectronics Reliability*, 2019.

[25] G. Brunetti, G. Campiti, et al. Cots devices for space missions in leo. *IEEE Access*, 2024.

[26] S. Chen and W. Guo. Auto-encoders in deep learning—a review with new perspectives. *Mathematics*, 2023.

[27] Y. Chen. Reram: History, status, and future. *IEEE Transactions on Electron Devices*, 2020.

[28] L. Chua. Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, 2003.

[29] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 2018.

[30] B. Denby, E. Ruppel, V. Singh, et al. Tartan artibeus: A batteryless, computational satellite research platform. In *Small Satellite Conference*, 2022.

[31] Y. Deng and J. Zhou. Architectures and optimization methods of flash memory based storage systems. *Journal of Systems Architecture*, 2011.

[32] Everspin Technologies. MR10Q010 Datasheet (Rev. 5.6). `https://www.everspin.com/sites/default/files/MR10Q010_Datasheet.pdf`, 2018.

[33] Fujitsu. MB85RS64V 64 K (8 K $\times$ 8) Bit SPI FRAM Datasheet. `https://www.fujitsu.com/uk/Images/MB85RS64V.pdf`, 2015.

[34] Fujitsu. MB85AS4MT 4 Mbit (512 K $\times$ 8) SPI FRAM Datasheet. `https://www.fujitsu.com/tw/Images/MB85AS4MT-DS501-00045-1v0-E.pdf`, 2021.

[35] S. Gerardin and A. Paccagnella. Present and future non-volatile memories for space. *IEEE Transactions on Nuclear Science*, 2010.

[36] C. Hafer, M. Von Thun, et al. Seu, set, and sefi test results of a hardened 16mbit mram device. In *IEEE Radiation Effects Data Workshop*, 2012.

[37] G. Harshvardhan, M. K. Gourisaria, et al. A comprehensive survey and analysis of generative models in machine learning. *Computer Science Review*, 2020.

[38] J. Heidecker, G. Allen, and D. Sheldon. Single event latchup (sel) and total ionizing dose (tid) of a 1 mbit magnetoresistive random access memory (mram). In *IEEE Radiation Effects Data Workshop*, 2010.

[39] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer. Fault injection techniques and tools. *Computer*, 1997.

[40] M. Islam, G. Chen, and S. Jin. An overview of neural network. *American Journal of Neural Networks and Applications*, 2019.

[41] A. Ju, H. Guo, et al. Analysis of ion-induced sefi and sel phenomena in 90 nm nor flash memory. *IEEE Transactions on Nuclear Science*, 2021.

[42] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[43] G. Korkian, D. León, et al. Single event upsets under proton, thermal, and fast neutron irradiation in emerging nonvolatile memories. *IEEE Access*, 2022.

[44] J. Lederer. Activation functions in artificial neural networks: A systematic overview. *arXiv preprint arXiv:2101.09957*, 2021.

[45] S.-K. Lu, H.-P. Li, et al. Fault-aware dependability enhancement techniques for phase change memory. *Journal of Electronic Testing*, 2021.

[46] H. Lyu, H. Zhang, et al. Research on single event effect test of a rram memory and space flight demonstration. *Microelectronics Reliability*, 2021.

[47] H. Madeira, D. Costa, and M. Vieira. On the emulation of software faults by software fault injection. In *Proceeding International Conference on Dependable Systems and Networks*, 2000.

[48] M. J. Marinella. Radiation effects in advanced and emerging nonvolatile memories. *IEEE Transactions on Nuclear Science*, 2021.

[49] E. Mathieu, P. Rosado, and M. Roser. Space exploration and satellites. *Our World in Data*, 2022.

[50] R. Natella, D. Cotroneo, and H. Madeira. Assessing dependability with software fault injection: A survey. *ACM Computing Surveys*, 2016.

[51] D. Nguyen and F. Irom. Radiation effects on mram. In *9th European Conference on Radiation and Its Effects on Components and Systems*, 2007.

[52] M. V. O'Bryan, K. A. LaBel, et al. Nasa goddard space flight center's compendium of recent single event effects results. In *IEEE Radiation Effects Data Workshop*, 2018.

[53] K. O'shea and R. Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[54] M. V. O'Bryan, K. A. LaBel, et al. Compendium of recent single event effects results for candidate spacecraft electronics for nasa. In *IEEE Radiation Effects Data Workshop*, 2008.

[55] G. Pagonis, V. Leon, et al. Increasing the fault tolerance of cots fpgas in space: Seu mitigation techniques on mpsoc. In *International Symposium on Applied Reconfigurable Computing*, 2023.

[56] V. M. Panaretos and Y. Zemel. Statistical aspects of wasserstein distances. *Annual Review of Statistics and Its Application*, 2019.

[57] P. Pavan, R. Bez, P. Olivo, and E. Zanoni. Flash memory cells-an overview. *Proceedings of the IEEE*, 1997.

[58] A. Poghosyan and A. Golkar. Cubesat evolution: Analyzing cubesat capabilities for conducting science missions. *Progress in Aerospace Sciences*, 2017.

[59] H. Quinn, P. Graham, et al. On-orbit results for the xilinx virtex-4 fpga. In *IEEE Radiation Effects Data Workshop*, 2012.

[60] D. Radaelli, H. Puchner, S. Wong, and S. Daniel. Investigation of multi-bit upsets in a 150 nm technology sram device. *IEEE Transactions on Nuclear Science*, 2005.

[61] F. Reghenzani, Z. Guo, and W. Fornaciari. Software fault tolerance in real-time systems: Identifying the future research questions. *ACM Comput. Surv.*, 2023.

[62] D. Selva and D. Krejci. A survey and assessment of the capabilities of cubesats for earth observation. *Acta Astronautica*, 2012.

[63] F. Semiconductor. Fram application note. `https://www.fujitsu.com/us/imagesgig5/FRAM_AppNote.pdf`, 2014.

[64] P. Shah and A. Lai. Cots in space: From novelty to necessity. In *35th Annual Small Satellite Conference*, 2021.

[65] J. Slaughter, R. Dave, et al. Fundamentals of mram technology. *Journal of superconductivity*, 2002.

[66] P. Stakem. Migration of an image classification algorithm to an onboard computer for downlink data reduction. *Journal of Aerospace Computing Information and Communication*, 2004.

[67] S. Tehrani, J. Slaughter, et al. Progress and outlook for mram technology. *IEEE Transactions on Magnetics*, 2002.

[68] T. Villela, C. A. Costa, et al. Towards the thousandth cubesat: A statistical overview. *International Journal of Aerospace Engineering*, 2019.

[69] Z. Wan, Y. Zhang, and H. He. Variational autoencoder based synthetic data generation for imbalanced learning. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017.

[70] C. Wang, C. Deng, and S. Wang. Imbalance-xgboost: leveraging weighted and focal

losses for binary label-imbalanced classification with xgboost. *Pattern Recognition Letters*, 2020.

[71] X. Wang, W. Li, Z. Luo, et al. A critical review on phase change materials (pcm) for sustainable and energy efficient building: Design, characteristic, performance and application. *Energy and buildings*, 2022.

[72] Y. Xie. *Emerging memory technologies: design, architecture, and applications*. Springer Science & Business Media, 2013.

[73] F. Zahoor, T. Z. A. Zulkifli, and F. A. Khanday. Resistive random access memory (rram): an overview of materials, switching mechanism, performance, multilevel cell (mlc) storage, modeling, and applications. *Nanoscale Research Letters*, 2020.

[74] M. Zanata, N. Wrachien, and A. Cester. Ionizing radiation effect on ferroelectric nonvolatile memories and its dependence on the irradiation temperature. *IEEE Transactions on Nuclear Science*, 2009.

[75] L. Zhang, W. Kang, Y. Zhang, et al. Channel modeling and reliability enhancement design techniques for stt-mram. In *IEEE Computer Society Annual Symposium on VLSI*, 2015.

[76] H. Ziade, R. A. Ayoubi, R. Velazco, et al. A survey on fault injection techniques. *Int. Arab J. Inf. Technol.*, 2004.

List of Figures

1.1 CubeSat external structure. 3

2.1 CubeSats common formats [5]. 8
2.2 Flash memory cell schema [10]. 12
2.3 FRAM cell schema [11]. 14
2.4 MRAM cell schema with parallel and anti-parallel configurations [75]. . . . 16
2.5 ReRAM cell schema and conducting filament creation [73]. 17
2.6 PCM cell schema and its equivalent circuit [45]. 19
2.7 GAN training flow [29]. 24
2.8 VAE training flow [45]. 26

4.1 CubeSat and onboard architecture. 32

5.1 Bit-flip mask creation. 39

6.1 Valid memory dumps per activation sessions fro each memory. 47
6.2 Evolution of bit flips percentage. 49
6.3 FRAM bit-flip masks examples. 51
6.4 Cumulative nuber of permanent blocks over activation sessions 54
6.5 FRAM block size distribution. 56
6.6 MRAM and ReRAM block size distribution over activation sessions. 57
6.7 Block size distribution across memories. 58
6.8 Block duration distribution across memories. 59
6.9 FRAM corruption heatmap. 61
6.10 MRAM partial corruption heatmap. 62
6.11 ReRAM partial corruption heatmap. 63

7.1 Fault generation workflow using the VAE. 71

List of Tables

4.1 Comparison of non-volatile memories. 33

6.1 Number of valid memory dumps for each memory 47
6.2 Mean and standard deviation of bit flips per memory type and sessions. . . 50
6.3 Permanent blocks features for each memory. 55

8.1 Mean and standard deviation for each fault model across memories. 78
8.2 Wasserstein distances for block size and block duration distributions for each fault model across memories. 79
8.3 Mean and standard deviation for machine learning model. 79

Acknowledgements

I would like to express my deepest gratitude to professor Luca Mottola for giving me the opportunity to be part of NesLAB and for his constant guidance and support throughout this journey. His passion for research and dedication have been truly inspiring and motivating to me.

I am also thankful to all the members of NesLAB for welcoming me so warmly and for sharing their knowledge and expertise. Working alongside you has been an incredible learning experience, and your kindness and teamwork have made my time in the lab both enriching and memorable.

A special thanks goes to my parents and family for their unconditional love and support. I am truly grateful to have you in my life, as your presence always gives me the courage to pursue my goals.

I would also like to thank my friends and my partner for always believing in me and standing by my side. Your constant support has been incredibly important to me, especially during difficult moments.