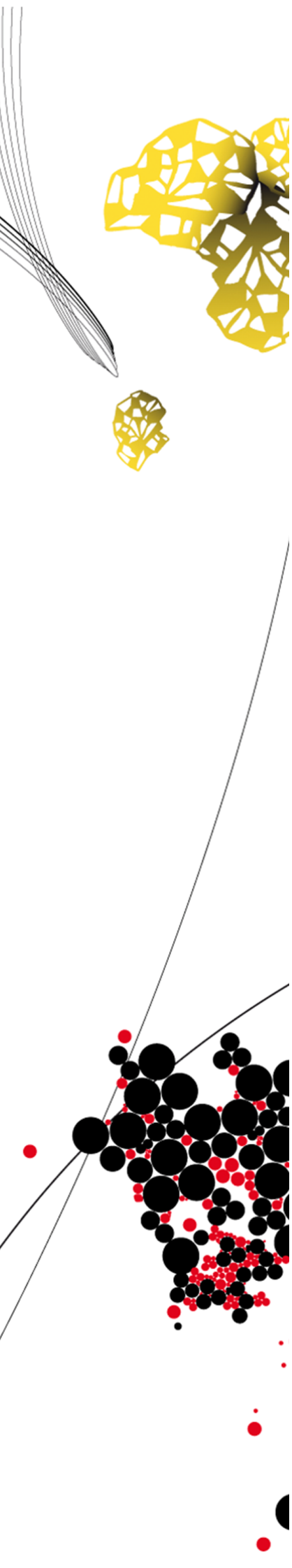


UNIVERSITY OF TWENTE.

**Faculty of Electrical Engineering,
Mathematics & Computer Science**

Improving Radiation Tolerance for Neuromorphic Processors

Wim, W.H. Nijsink

M.Sc. Thesis
August 2025

Supervisors:
dr. ir. M. Ottavi
ir. B. Endres Forlin
dr. ir. A. Yousefzadeh
dr. ir. A. Chiumento

Computer Architecture for Embedded Systems
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Preface

When looking around for a research topic for my thesis, I got in contact with the Computer Architecture for Embedded Systems (CAES) research group. The challenge of doing a very diverse and multidisciplinary project, including exploring neuromorphic architectures, creating an FPGA design, and doing research on radiation effects, attracted my attention, and that is why I started doing this research. The topic of using online learning to withstand radiation has not been well explored for neuromorphic architectures, which offers a good opportunity to develop new scientific insights, which also motivates me to do this research.

I had a lot of new fields and material to study; not only was the field of neuromorphic AI models new for me, but also the field of radiation testing. I want to thank my supervisors for guiding me through all this during this research project. I want to thank Bruno for helping me prepare for the radiation tests and introducing me to the science of radiation testing. Our research trip to the UK was also a very nice experience, and I am glad to have been part of that. I would like to thank Amir for his explanation of neuromorphic architectures and for his helpful questions and guidance with finding the research direction. I appreciated Marco's critical questions, which not only helped shape the thesis, but also provided valuable insights into presenting and interpreting the collected data.

I look back on a fruitful collaboration with my supervisors, and I am proud to present my research results in this thesis report.

Summary

This thesis investigated the radiation tolerance of neuromorphic architectures. This was triggered by the findings from the existing literature that neuromorphic architectures have good radiation tolerance. To improve on this radiation tolerance, we introduced online learning, an Spike-Dependent Synaptic Plasticity (SDSP) learning mechanism that runs unsupervised on the system and has the task of counteracting radiation influences.

To validate our approach, we needed to expose our architecture to radiation. To make a test target for this, we needed an FPGA, for which we selected a flash-based FPGA. The device we picked was the PolarFire® SoC FPGA Icicle Kit. We also had to choose an open-source neuromorphic architecture to place on that FPGA; for this, we did a design space exploration and selected the Online-learning Digital spiking Neuromorphic processor (ODIN) architecture due to its learning capabilities and its relatively small size. We then created monitoring logic around the neuromorphic implementation and created a test setup. We also had to train the neuromorphic architecture and configure the online learning parameters. We selected the MNIST dataset for training. The task of the neuromorphic architecture was to classify those images. The initial model training was done with Python, the resulting weights of this training were put on the FPGA and that gave us the starting point for radiation testing. We created a UART connection to the FPGA to configure the architecture, set up the weights, and also to send the images and retrieve the classification output.

With our neuromorphic architecture setup on the FPGA device, we were able to test the architecture against radiation. We started with radiation beam testing at the ChipIR testing facility at Appleton Laboratory in the UK. This radiation beam allowed us to collect data about the radiation resistance of our system. We expanded the data collection by running a simulation based on the measured fault rate to get more information on radiation effects over a longer time period.

The ODIN architecture showed good radiation resistance without additional learning, but with added learning, the accuracy in the shorter runs decreased instead of increased. We ran additional simulations and found that over a longer period of time learning will have positive effects. We ran simulations until we had a completely failed system without learning. We did not see complete system failure with learning

enabled, so this means that learning increases the time it will take for the system to completely fail by at least 175%. Therefore, while learning does not significantly contribute to preserving the system's initial high level of accuracy against radiation effects, it is highly effective in substantially delaying the complete failure of the system.

Contents

List of acronyms

FPGA	Field-Programmable Gate Array
ODIN	Online-learning Digital spiking Neuromorphic processor
SEU	Single Event Upset
SEUs	Single Event Upsets
SET	Single Event Transient
SETs	Single Event Transients
SEFI	Single Event Functional Interrupt
SEL	Single Event Latchup
SEDR	Single Event Dielectric Rupture
SDSP	Spike-Dependent Synaptic Plasticity
STDP	Spike-Timing-Dependent Plasticity
DUT	Device under test
SUT	System under Test
UART	Universal Asynchronous Receiver-Transmitter
TMR	Triple Modular Redundancy
SPI	Serial Peripheral Interface
AER	Address Event Representation
MSB	Most Significant Bit
LSB	Least significant Bit
MTTF	Mean Time To Failure
MTBF	Mean Time Between Failure
FIT	Failures in Time

Chapter 1

Introduction

With the increasing demand for energy-efficient and intelligent [27] computing devices, neuromorphic processors have emerged. The first neuromorphic system, which incorporates basic elements such as synapses and neurons, was developed in 2006 [9]. Neuromorphic processors offer several advantages, most notably their energy efficiency and suitability for real-time and low-power applications. Neuromorphic processors work by mimicking the neural architecture of the human brain [24], which is known to be energy efficient [17].

A neuromorphic processor can be used in different critical environments such as automotive, space, medical devices, and nuclear facilities [24]. In deployment environments such as space or high-altitude aviation, neuromorphic processors are subjected to external influences such as radiation, which may compromise their reliability. This study specifically focuses on neutron radiation, which occurs both in atmospheric and space environments.

Existing studies indicate that neuromorphic processors have a higher radiation tolerance compared to CPU and GPU implementations [21]. Choices about the architecture of the neuromorphic system significantly influence radiation tolerance. For example, the accuracy of neuromorphic systems for an image classification task did not change significantly when radiation was applied and caused Single Event Upsets (SEUs) [7]. In addition, the application of error correction techniques also improves reliability [14]. Moreover, using redundancy helps ensure that operations continue when the processor is under radiation [3]. Additionally, self-healing and adaptive mechanisms can respond to radiation-induced changes that will improve radiation tolerance [2].

To improve radiation resistance, this research will focus only on optimizing the architecture itself, not applying existing and proven error correction techniques or redundancy. This study investigates the use of Spike-Dependent Synaptic Plasticity (SDSP) learning to mitigate the impact of radiation on inference accuracy and architectural robustness. The reason for this is to see the actual effects of improve-

ments in neuromorphic architecture without the confounding effects of established hardware reliability techniques such as error correction and redundancy.

This results in the following main research question:

To what degree and in what manner can neuromorphic architectures endure radiation exposure? To answer that question, we aim to answer the following sub-questions:

- To what extent does the implementation of online learning during inference enhance the radiation resilience of neuromorphic architectures?
  - Can online learning mechanisms serve as effective substitutes for traditional error correction methods in neuromorphic architectures subjected to radiation exposure?
  - How do the spatial, power, and performance requirements for the implementation of online learning compare with those for traditional error correction mechanisms in neuromorphic chips?
- Which parts of a neuromorphic architecture are the most vulnerable to radiation influences?
  - How can we inject faults into the hardware?
  - How to monitor the number of faults occurring in the memory areas for the weights and neurons states?
  - How vulnerable are spikes data-structures to faults?
- How can radiation tolerance be improved with training the model in a different way?
  - Is quantization a feasible technique to improve radiation tolerance?
  - Is a sparser network more resistant to radiation?

The remainder of this thesis is organized to support the progression from a background understanding to an experimental validation. Chapter 2 provides background information; Chapter 3 reviews related work; Chapter 4 outlines the methodology; Chapter 5 presents the results, followed by a discussion in Chapter 6. Chapter 7 concludes the thesis with a summary of findings and future directions.

Chapter 2

Background

Before diving into the related work and the methodology for the research, the key concepts related to this research are presented in this chapter. In the first section, a brief overview of how neuromorphic computing works is provided. In this overview, we show the basic concepts and explain how they work together to build a neuromorphic system. The next section provides an exploration of the fundamental concepts of radiation, tailored to the emphasis this study places on radiation. This discussion elaborates on the necessary definitions and equations essential for understanding the processes involved in performing radiation tests.

2.1 Neuromorphic computing

A neuromorphic processor consists of neurons and synapses. Those neurons form a network where the weights indicate the connection strengths. Within this network, the spikes travel from neurons to synapses. Those parts require some explanation, so we will go over them one by one. A simple schematic of how a neuromorphic processor works is shown in Figure 2.1

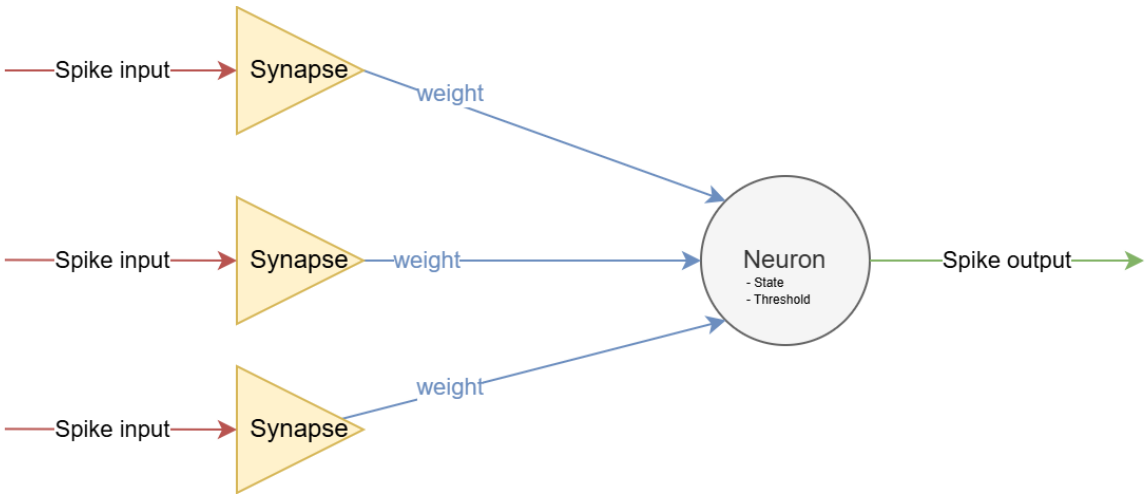


Figure 2.1: A simple overview of a neuromorphic processor

Neurons A neuron forms a part of the neuromorphic processor, the neurons are the heart of the chip. The intelligence of the neuromorphic processor lies in the network and the strength of the connections among neurons. Neurons maintain an internal state and process incoming spikes. They also generate outgoing spikes based on their state. They can be set up to fire, meaning that they send a spike once a certain threshold is reached. The number of neurons can vary and can be adjusted for the intended application of the neural network.

Spikes In a neuromorphic processor the input signals are spikes, for a digital processor, those spikes are either ‘0‘ or ‘1‘.The inputs are connected to synapses, which are a kind of input receiver. Those synapses have connections to neurons and carry events throughout the system. There are not only input spikes, but also output spikes, which are created by the neurons. The output spikes can travel through the network like the input spikes. They have two purposes, they can travel to other synapses and cause other neurons to spike, and/or they can be used to do classification. Classification can for example be based on the firing frequency or on the count of output spikes per neuron.

Weights For each connection from a synapse to a neuron, a weight is configured, the weight indicates the strength of the connection. The spike signal multiplied by the weight then activates a neuron. Each neuron has a threshold, and for a basic implementation the sum of incoming spikes can be used to determine the state. When this state exceeds the threshold, the neuron itself fires. The firing of a neuron means that the neuron sends out a spike signal. This spike signal from the neuron is

then distributed through the synapse network. The output spike signal is captured by the synapses, which will bring the spike signal as input back to one or more neurons.

Synapses Synapses are part of the connection that receives spikes; we can see the synapses as the interface of the neurons. The synapses pass the spike to the neuron and indicate the significance of the spike using a weight. Those synapse weights are an important part of the neural network and they need to be trained to make the network do something meaningful.

Examples The advancement of neuromorphic computing technology has culminated in the development of substantial chips with capabilities to execute neuromorphic processing. Numerous neuromorphic computing chips currently exist that implement these processes, including the Intel Loihi, TrueNorth, NeuronFlow, SpiNNaker, and Epiphany chip [19].

2.2 Spike-Dependent Synaptic Plasticity (SDSP)

A key concept to comprehend is Spike-Dependent Synaptic Plasticity (SDSP), which will be used as the foundation for the learning approach in this research. Initially, it is good to explain two closely related terms. The notion of Spike-Timing-Dependent Plasticity (STDP)[15] bears resemblance to SDSP, yet it exhibits distinct differences.

STDP The Spike-Timing-Dependent Plasticity (STDP) is dependent upon the precise timing between pre-synaptic and postsynaptic spikes [13]. STDP encodes temporal causality by purely relating spike activities through time differences. This form of timing-dependent plasticity has biological roots as it emulates neural mechanisms in the brain. The weight is adjusted according to the order in which the presynaptic spike and the postsynaptic spike occur. The presynaptic spike is expected to be earlier than the postsynaptic state, if that is the case the weight is incremented. If the presynaptic spike comes after the postsynaptic spike then the weight is decreased, because the neuron fires without the need of this presynaptic spike so the relation is not relevant to the spike output.

SDSP Conversely, Spike-Dependent Synaptic Plasticity (SDSP) represents a broader categorization of relating spike activities [13], which does not rely on the very strict timing order. This approach combines postsynaptic states with the spike timing and simplifies computational tasks by doing the weight update based on the postsynaptic state crossing the configured threshold; this removes the need of delay lines or

timing measurement and thus removes stringent timing requirements. This methodology is advantageous in neuromorphic architectures due to its hardware-friendly nature, which facilitates easier implementation and demands less computational power [5].

2.3 Radiation testing

Radiation causes problems in the reliability, and thus the accuracy, of computing systems because radiation causes disruptions on the hardware level. Those disruptions cause faults to occur. For specific applications of devices, such as usage in outer space, the device is exposed to much more radiation than on Earth [4]. To avoid unexpected issues that arise from radiation on the device, preliminary radiation testing is the way to go to identify early how capable the hardware is of handling radiation.

Fault types When addressing faults, the types to consider when radiation testing on a FPGA are Single Event Functional Interrupt (SEFI), Single Event Latchup (SEL), Single Event Dielectric Rupture (SEDR), Single Event Upset (SEU) and Single Event Transient (SET) [20]. We will explain those faults in a bit more detail below.

A Single Event Functional Interrupt (SEFI) occurs in a control register causing changes in the operating characteristics of a circuit. The system may hang or freeze and often requires a reset or a power cycle to recover.

A Single Event Latchup (SEL) is a hard error, which means that there is a short circuit in the circuit caused by radiation, which can most of the times be resolved by power cycling the device. If not resolved quickly enough, the current can increase to much, causing overheating and possible permanent damage.

A Single Event Dielectric Rupture (SEDR) means that physical damage is done to the FPGA, a current is flowing through the dielectric, this happens when the radiation lowers the resistance of the dielectric. This is very rare, but it leaves permanent damage to the chip.

Single Event Upsets (SEUs) occur when radiation affects a memory cell, causing a change in its stored value, a phenomenon known as bit flips. In contrast, Single Event Transients (SETs) impact combinatorial logic, leading to erroneous data being stored in memory. Although both events can occur, SEUs are significantly more common. [26]

Radiation metrics Radiation tolerance is a metric that indicates how well the system or device can handle radiation. To measure tolerance, we need to introduce faults and monitor how the system reacts on those faults. This is why the study on radiation effects requires radiation testing. Testing the radiation tolerance of a device requires a radiation source. From this source radiation can be emitted on the device. The device needs to be monitored to collect data that should show the effects of radiation. For formulas for radiation-related calculations, we rely on the existing terminology [10].

Radiation is measured in flux denoted with the symbol Φ.

$$\Phi = \frac{\text{particles}}{\text{cm}^2 \cdot \text{s}} \tag{2.1}$$

The total amount of radiation received is called fluence, denoted with the symbol Ψ. We integrate the flux over time to get the fluence.

$$\Psi = \int \Phi(t)\, dt \tag{2.2}$$

To gain insight into the vulnerable area of the device to radiation effects, we calculate the cross section, which will be denoted as σ.

$$\sigma = \frac{\text{Number of observed errors}}{\Psi} \tag{2.3}$$

To calculate the failure rate λ we need to multiply the cross section by the flux:

$$\lambda = \Phi \cdot \sigma \tag{2.4}$$

With the failure rate we can calculate the Mean Time To Failure (MTTF).

$$\text{MTTF} = \frac{1}{\lambda} = \frac{1}{\sigma \cdot \Phi} \tag{2.5}$$

To calculate the Failures in Time (FIT), which gives us the number of failures per billion hours of operation, we need the following formula:

$$\text{FIT} = \frac{3600 \cdot 10^9}{\text{MTTF}} = 3.6 \cdot 10^{12} \cdot \sigma \cdot \Phi \tag{2.6}$$

We see two parameters in this formula, the cross section and flux, if the device becomes more vulnerable to radiation, the cross section Φ increases. When there is more radiation the flux Φ increases. Any increase in those parameters increases the number of Failures in Time (FIT) as well.

2.4 Radiation and FPGA technology

To comprehend the decisions related to the selection of an FPGA type, it is imperative to compare the various available FPGA types and assess their performance under conditions of radiation exposure. In order to concentrate on fault detection within the architecture, it is of important to select an FPGA technology that facilitates this process [23]. For this, we consider FPGAs based on SRAM, Flash, and antifuse [6].

An SRAM-based FPGA employs volatile memory for configuration storage, necessitating re-programming upon each device power-up. This type of memory, housed within SRAM, exhibits sensitivity to radiation effects. Conversely, a Flash-based FPGA utilizes flash to maintain its configuration, leveraging nonvolatile memory, thereby obviating the requirement of reprogramming upon device power cycles. Flash cells employ floating-gate transistors, which are resistant to radiation exposure [10]. Antifuse-based FPGAs also employ nonvolatile memory; however, they can only be programmed once, making them less suitable for prototyping purposes. The configuration implemented in antifuse-based FPGAs is permanent and immune to radiation effects.

One of the primary benefits of Flash-based FPGAs is the resilience of their configuration memory in the face of radiation exposure, an attribute that facilitates testing without risking the integrity of the configuration due to potential corruption. As a result, the decision was made to utilize a Flash-based FPGA.

The PolarFire® SoC FPGA Icicle Kit[1] was selected for our experimentation due to its compliance with the criterion of being a flash FPGA. Its availability at the university facilitated the immediate beginning of our work. Furthermore, the widespread availability of the board simplified the configuration process for radiation testing purposes. Furthermore, the substantial existing expertise with these boards within the university community reduced the likelihood of encountering non-contributory challenges during the research process.

[1]We used the ES (engineering sample) version, see https://www.microchip.com/en-us/development-tool/MPFS-ICICLE-KIT-ES

Chapter 3

Related work

To make well-informed decisions ahead of conducting research, it is crucial to examine the existing body of literature to circumvent redundant efforts. Our investigation commences with an exploration of current neuromorphic architectures, seeking an architecture that aligns with our objectives. Utilizing a proven, functional neuromorphic architecture as a foundation for our study allows us to build upon established designs rather than redevelop them entirely. Additionally, we explore related literature concerning the radiation tolerance of neuromorphic architectures. Such studies provide valuable insights into the existing knowledge gaps that our research aims to address.

3.1 Neuromorphic processor architectures

In order to gain access to an established architecture, a design space exploration was conducted to compare various open-source spiking neural network architectures [22]. The evaluation includes the following architectures: ODIN [12], ReckON [11], RANC [16], SNE [8], and OpenSpike [18], in addition to two smaller architectures; 'A Lightweight Spiking Neural Network Accelerator [Google Shuttle]' and 'SNN ASIC accelerator [Google Shuttle]', for which no publications are available.

In Table 3.1 an overview is given of the criteria used to select the architecture for this research.

The initial inquiry was whether the architecture was compatible with the target Field-Programmable Gate Array (FPGA)[1]. In response, all architectures were considered suitable. Otherwise, we might have to consider a different target FPGA.

Then we tried all architectures to see if they will cause compatibility errors with the PolarFire® FPGA and the corresponding Libero toolchain; in this test the SNE architecture failed, so we no longer consider SNE a valid option. The usage of in-

[1] See section 2.4 for explanation about the FPGA target selection

Architecture	FPGA Compatible	Libero Compatible	Online Learning	Excluded Reason
ODIN	Yes	Yes	Yes	—
ReckON	Yes	Yes	Yes	More complex than ODIN
RANC	Yes	Yes	No	No online learning
OpenSpike	Yes	Yes	No	No online learning
SNE	Yes	No	No	Incompatible with Libero toolchain
Lightweight SNN Accelerator	Yes	Yes	No	Too simple, lacks documentation
SNN ASIC Accelerator	Yes	Yes	No	Too simple, lacks documentation

Table 3.1: Comparison of selected open-source SNN architectures for FPGA implementation

compatible Verilog syntax makes it hard to move the implementation over to different boards, so this is a knock-out criterion.

The architectures referred to as 'A Lightweight Spiking Neural Network Accelerator' [2] and 'Spiking Neural Network ASIC Accelerator' [3] were excluded from consideration for two primary reasons. Firstly, their very simple design renders them insufficiently robust to handle standard datasets available for neuromorphic computing. Secondly, the absence of accompanying articles or documentation provided insufficient information to ascertain their viability for our purposes.

This analysis focuses on the four remaining architectures, which require a comprehensive comparison. These architectures can be categorized into two groups: those with online learning capabilities and those without. Notably, ODIN and ReckOn support online learning functionalities, whereas RANC and OpenSpike lack such features. Our intention to examine the impact of faults [4] in the context of online learning effectively excludes the RANC and OpenSpike architectures from our consideration. Consequently, the decision is limited to selecting between ODIN and ReckOn. To minimize complexity and streamline the research process, the ODIN architecture is chosen for its simplicity.

So we selected ODIN [12] due to the ability to do online learning, the small memory size, and the lowest complexity.

[2] https://github.com/jeshraghian/snn-accelerator
[3] https://github.com/pengzhouzp/wrapped_snn_network
[4] See section 4.1 for more details on the type of faults we target

ODIN architecture The readily available open-source ODIN architecture is depicted in Figure 3.1, and is implemented using Verilog. In Figure 3.1 the Verilog modules are presented. Each module is designated with a distinct responsibility, ensuring that the architecture remains comprehensible and accessible for modifications. The top module, called ODIN, combines all the required parts and connects them to each other.

We have two modules for communication: the AER_OUT module is an interface directly to the neuromorphic chip and can also be used to connect multiple chips. The Address Event Representation (AER) interface is also used for spike signal communication. For external communication, we have the SPI_Slave module which provides logic to process incoming Serial Peripheral Interface (SPI) communication, the available functionality for the SPI interface consists of reading/writing to both the neuron and the synapse memory. The SPI interface also provides the functionality to configure the ODIN architecture, such as enabling learning or configuring the SPI.

The controller module controls the ODIN architecture; this is done by handling the communication and organizing the spike flow through the network.

The synaptic core module contains the synapses and contains the update logic for the synapses, as well as the learning logic required for SDSP learning.

The neuron core module holds the neurons and the logic for updating the neuron data.

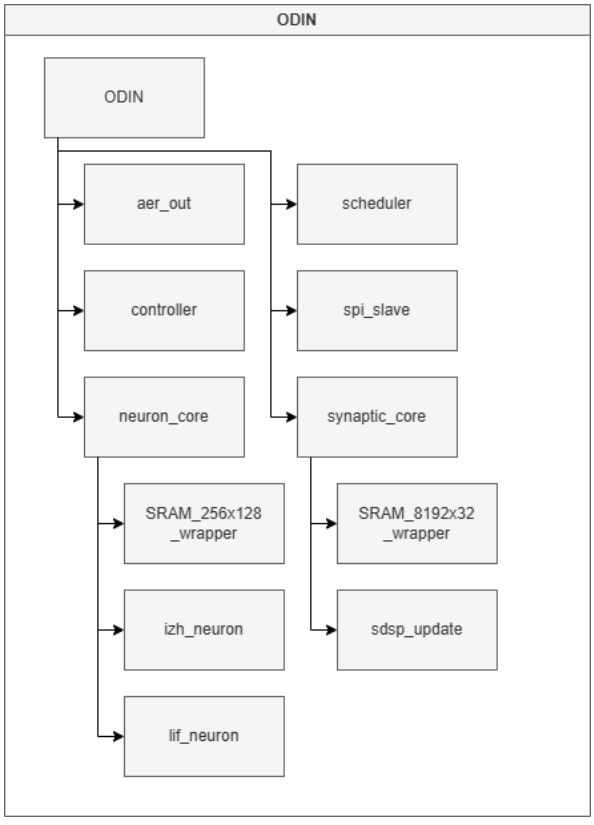


Figure 3.1: The ODIN architecture

3.2 Radiation on neuromorphic architectures

Some literature on radiation testing on neuromorphic architectures is available for this study. This chapter will display several articles which contain work that is already done in radiation experimentation with neuromorphic architectures.

Evidence indicates that single-event disturbances exhibit negligible effects on image classification within neuromorphic computing architectures [7]. The research presented by the authors in [7] employed a TrueNorth chip for executing an image classification task utilizing the MNIST dataset, which comprises 10,000 test images. The pretrained network achieves an accuracy of 98.82%. This high level of accuracy can be attributed to the chip's substantial size; the TrueNorth chip is equipped with 4096 cores and 428Mb of SRAM [1]. Their experimental procedure involved the use of protons from a 4MeV source with a variable flux range 10^5 to 10^{10}protons/cm^{2}s.

A noteworthy finding from the radiation tests executed in the aforementioned study was that the overall accuracy remained unaffected. However, some classification inaccuracies were recorded, wherein certain erroneously classified images were rectified while some accurately classified images faced misclassification. The architectural evaluations were conducted for fluence levels of $5.6 \cdot 10^7$cm^{-2}.

A survey paper [21] provides a comparative analysis of neuromorphic architectures against traditional CPU and GPU implementations. The survey shows that neuromorphic hardware has 5 to 10 times longer Mean Time Between Failure (MTBF) compared to CPU and GPU implementations [21].

A critical factor influencing radiation tolerance is the architecture of neuromorphic systems. Within the architecture, four domains are highlighted as influencing radiation tolerance: device technology and system design, error correction technologies, redundancy, and adaptive, self-healing mechanisms. As demonstrated in [7], the device technology inherent in neuromorphic architectures already exhibits considerable radiation resistance. Moreover, hardware platforms play a role in radiation tolerance, with CPU and GPU architectures being identified as more susceptible, according to [21]. The implementation of error correction codes, as evidenced by Hassan [14], aids in rectifying radiation-induced errors, thereby enhancing reliability. This novel error correction implementation also contributes to energy savings of up to 31% while maintaining performance metrics comparable to existing techniques. Another strategy to augment error resistance in architecture is the utilization of checker neurons [3], which approximate the functions of hidden layers to mitigate unexpected anomalies and pronounced outliers.

Another example of radiation testing on neuromorphic chips is represented by the evaluation conducted on the Intel Loihi chip [25]. This chip was tested without the implementation of any methodology aimed at improving radiation tolerance,

which resulted in finding out that the chip is susceptible to malfunctions. The chip experienced operational freezes, and demonstrated latch-up phenomena when it was exposed to proton radiation [25].

Chapter 4

Methodology

We want to prove that online learning improves radiation tolerance for neuromorphic architectures. To do that, we need to define a fault model, select a neuromorphic architecture, put that on an FPGA and then put that to a test bench which allows us to monitor the behavior under radiation.

A robust methodology to test our hypothesis and gather results is required. We align our methodology with an already proven radiation testing methodology [10]. To do this, we need to implement a neuromorphic architecture on an FPGA. An FPGA provides the capability for dynamic modification and adaptation of the architecture, in addition to facilitating the integration of supplementary logic for configuration and control. The implementation of advanced error detection mechanisms is further enhanced by the adaptable FPGA platform. Consequently, it is necessary to select a neuromorphic architecture to deploy on the FPGA, which serves as a framework for the examination of radiation resistance. To evaluate radiation resistance, it is crucial to monitor the effects of faults introduced via fault injection. These faults can be manually injected or can be induced in the hardware by utilizing a radiation beam. To test with a radiation beam we traveled to Rutherford Appleton Laboratory [1] in the UK, for manual fault injection we have to set up a fault injection mechanism. By analyzing the consequences of these faults, the level of radiation tolerance will be determined.

4.1 Fault model

We are looking for faults to occur in the implemented neuromorphic architecture, but we are not interested in all possible faults that radiation could cause on the hardware. The basic idea is that we only want to focus on the neuromorphic implementation as if it has been taped out, so we do not want to be bothered with FPGA specific

[1] See https://www.ukri.org/who-we-are/stfc/facilities/rutherford-appleton-laboratory

radiation issues.

To effectively address the emerging faults within the architecture, it is essential to employ FPGA technology that enables such a focus. Hence, we have chosen the PolarFire® SoC FPGA Icicle Kit[2].

The various potential faults are presented in Table 4.1. Within the table, we identify the faults of interest using the ✓symbol.

Faults classified under category 1 are addressed by the application of Triple Modular Redundancy (TMR). Furthermore, TMR is implemented for issues 2.4 and 3.5. This methodological approach allows for a concentrated effort on faults inherent to the ODIN architecture as opposed to peripheral system testing. In the event that faults persist despite the application of TMR, a device power cycle becomes necessary. Within our architectural framework, occurrences of SEU may manifest within memory regions allocated for weights and neuron states. Simultaneously, SET possess the capability to interfere with model processing during interference, potentially leading to the generation of erroneous spikes. Although both fault types are possible under radiation exposure, the principal focus is on SEU, due to the relative ease of measurement and higher incidence rate of this fault type. To quantify the SEU, memory readings are conducted pre- and post-radiation exposure, facilitating a comparative analysis to determine the incidence of SEU.

[2]See section 2.4 for background information about this decision

Category	Subcategory	Description	Interesting
1. SEFIs / SELs / SEDRs (Hardware failures)	1.1 FPGA freezing or crashing	Complete system lock-up due to latchup or functional interrupt.	✗
	1.2 UART communication failing	Loss or corruption of serial communication due to peripheral failure.	✗
	1.3 Ethernet communication failing	Packet drop, link loss or corruption during high-speed data transfer.	✗
	1.4 Internal SPI bus failing	Bus contention or transaction failure between components.	✗
	1.5 Clock issues	PLL failure or clock glitches causing system desynchronization.	✗
	1.6 Configuration corruption	Corruption of configuration bits due to radiation or voltage spikes, leading to unpredictable behavior.	✗
	1.7 Watchdog timeout / system reset	Failure to service watchdog timer due to logic hang or stalled software.	✗
2. SEUs (Bitflips in sequential elements)	2.1 Neuron memory bitflips	Corruption of internal neuron state (e.g., membrane potential).	✓
	2.2 Synapse memory bitflips	Faulty synaptic weights or dropped connections.	✓
	2.3 ODIN configuration register bitflips	Alteration of threshold, timing, or routing parameters.	✓
	2.4 Additional control logic bitflips	State machine corruption in peripheral or control logic.	✗
3. SETs (Glitches in combinational logic)	3.1 Neuron processing	Invalid or missed spikes due to transient in decision logic.	✓
	3.2 Synapse computation	Glitch in applying weight logic.	✓
	3.3 Spike routing	Misrouting or dropped spikes from transient address error.	✓
	3.4 Learning	Faulty adaptation due to incorrect timing or update condition.	✓
	3.5 Control and scheduling logic	Transient control signal error affecting global behavior.	✗

Table 4.1: Fault Model for Radiation Testing of Flash FPGA-Based Neuromorphic Architecture

4.2 Neuromorphic architecture

The ODIN architecture has been implemented on a flash-based FPGA. When radiation hits the board, we want to be sure that radiation does not influence the configuration stored on the FPGA. To mitigate that risk we need a flash FPGA to resolve the risk of a corrupted gate configuration when radiation is applied.

To make radiation testing possible and to be able to control the ODIN architecture, we created additional Verilog modules and also made a few changes to the ODIN implementation.

Verilog implementation The ODIN architecture is open source and available as Verilog code. To expand the architecture, we decided to use Verilog as well so that we can easily connect the new code to the existing Verilog code.

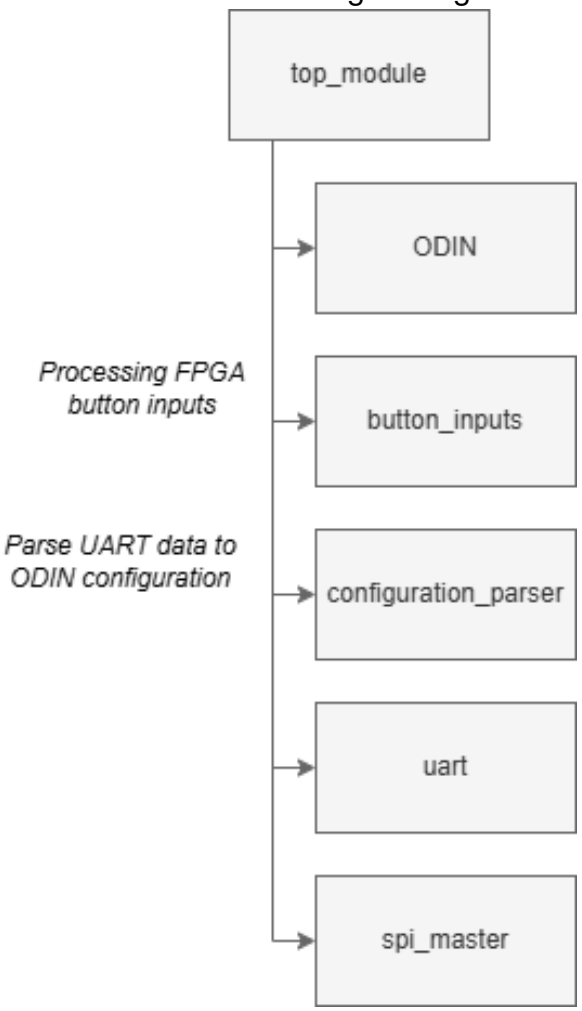


Figure 4.1: Additions to the ODIN architecture

We needed a way to facilitate interacting with the ODIN architecture. For that additional Verilog modules are created. Those modules are shown in Figure 4.1

ODIN The ODIN module is the modified top module of the ODIN architecture. This module is changed to make reading the configuration possible; also some minor fixes and changes are also implemented. This includes additional logic to read the configuration from the ODIN architecture, so that we can validate that setting the configuration is done correctly. It also includes removing some unused signals, adding some macro functions like $(*syn\_ramstyle = "rw\_check"*)$ and expanding the reset functionalities.

Button inputs For debugging purposes, we use the hardware buttons on the FPGA, to handle those inputs we created a module button_inputs.

UART interface In order to facilitate external connectivity to the ODIN architecture, an Universal Asynchronous Receiver-Transmitter (UART) interface has been integrated. This UART interface employs a simple protocol to transmit commands and data to the ODIN architecture. The suite of commands that have been implemented include the following:

- Reset command
- Image command (part 1)
- Image command (part 2)
- Configuration command
- Write neuron memory command
- Read neuron memory command
- Write synapse memory command
- Read synapse memory command

The reset command functions equivalently to a power cycle or the activation of the reset button, by resetting all memory data to zero and reverting all state machines to their default states. The image command is split into two parts due to the limitation of the RadHelper tool which allows a maximum message length of 255 bytes, whereas a complete image requires 256 bytes; further details about RadHelper can be found in Appendix B. Due to this limitation, the image commands are divided into two segments, each containing a half image. The configuration command sets all the configuration bits for the ODIN architecture. The configuration parameters include enabling or disabling SPI access, and also include possibilities to modify behavioral settings, like enabling or disabling learning. The write and read commands

are designed for the configuration of memory and the retrieval of memory data, respectively. The write commands specify both the memory address and the corresponding data to be stored, whereas the read commands specify only the address to be accessed. For each command, with the exception of the reset command and the initial part of the image command, a response is sent back via the UART interface, which contains either the requested data or the result of the data processing.

Configuration parser The configuration parser is a very simple module that converts the bits from the configuration command into meaningful signals that are then sent to the ODIN architecture over the internal SPI interface.

SPI master The SPI master controls the SPI slave. The SPI master allows us to write data towards the ODIN architecture. The UART interface uses this SPI master to bring the UART commands to the ODIN architecture for processing.

Bitstream generation We use the PolarFire® SoC FPGA Icicle Kit[3] The implementation was done in Verilog, because the open-source ODIN architecture was written in Verilog as well.

We generate the bitstream from the Verilog code, using Libero[4], the default toolchain supplied by the board manufacturer.

The final bitstream does not fully utilize the FPGA, so while designing the control logic around the ODIN architecture, we did not have to be afraid of making it impossible to fit. The resource usage of the chip is shown in Table 4.2. For more details on resource usage, see section 5.4.

Type	Used	Total	Percentage
4LUT	75,174	254,196	29.57%
DFF	59,712	254,196	23.49%
I/O Register	0	142	0.00%
Logic Element	110,610	254,196	43.51%

Table 4.2: Resource Usage Statistics

[3]We used the ES (engineering sample) version, see https://www.microchip.com/en-us/development-tool/MPFS-ICICLE-KIT-ES

[4]Libero SoC v2024.2 with a silver (free) license, see https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/fpga/libero-software-later-versions

4.3 Validation

To validate that we have a working system, we use the ODIN architecture for image classification. Validation is done by running interference on images and determining the class to which the image corresponds. For this image classification, we used the MNIST data set.

MNIST The ODIN framework is trained using 60,000 images from the MNIST training data set. For offline network training, we used Python, leveraging the tensorflow.keras framework. See Listing 4.1 for reference.

For the model, a python representation of the ODIN architecture was used, so only one fully connected layer was used. The layer contains 10 neurons, which correspond to the 10 classes, the digits 0-9.

Listing 4.1: Python Model Definition

```
model = Sequential([
            Flatten(input_shape=(16, 16)),
            Dense(10),
            Softmax()
        ])
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x = x_train,
          y = y_train,
          epochs=200,
          validation_data=(x_test, y_test))
```

Before quantization, the model reached an accuracy of 92.5%, after training for 200 epochs, after quantization to only 3 bit weights, we have an accuracy of 82.5%. The model weights obtained in this way are used as the starting point of the ODIN architecture on the FPGA.

We started out with the regular MNIST dataset but we made some modifications to this dataset. Those modifications are needed for the ODIN architecture.

To make the model compatible with the ODIN architecture we applied the following steps to the MNIST dataset:

- Resizing (change the image size from 28x28 to 16x16)
- De-skewing (making the images less variable in their rotation)

- Thresholding (limiting the values, which is needed for quantization of the images)
- Convert image to spikes (needed for the neuromorphic architecture)

First of all, we only have 256 input synapses, so the image should not be larger than 256 pixels. To get the image to that size we resized the images to 16x16 pixels. After resizing, we de-skewed the images and applied a threshold to the pixel values. The last step before being able to send the images to the ODIN architecture is to convert the pixel values into spikes. We use a spike representation where the spike frequency is determined by the brightness of the pixel, so higher pixel values will result in more spikes for that pixel.

A visual representation of the conversion of an MNIST image to a spike representation is shown in Figure 4.2.

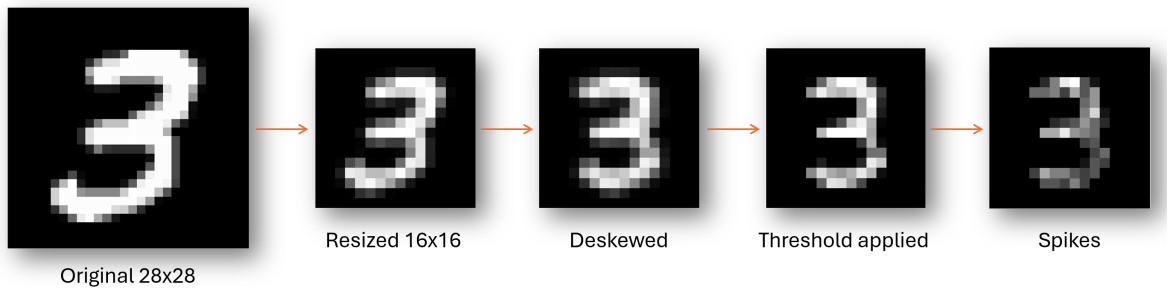


Figure 4.2: The preprocessing steps applied to the MNIST images

4.4 Test setup

In order to understand how the neuromorphic architecture performs when exposed to radiation, we visited Rutherford Appleton Laboratory[5]. At this facility, we carried out beam experiments, wherein the FPGA board, preprogrammed with our neuromorphic architecture, was subjected to the neuron beam. To ensure a highly stable and radiation-resistant environment surrounding the System under Test (SUT), we employed existing radiation hardening techniques. Subsequently, the device needs to be physically placed under the beam. To obtain results, it is essential that we establish communication with the device. The final section of this chapter discusses the tests we conducted and how we optimized the use of the time window.

Radiation hardening To shield the Device under test (DUT) from radiation as effectively as possible while exposing the SUT, we oriented the board to concentrate

[5]Rutherford Appleton Laboratory, Harwell Oxford, Didcot OX11 0QX, United Kingdom

the radiation on the section containing the FPGA chip, where the SUT is located. The other sections of the board should receive minimal radiation to prevent interference that might obscure issues within the SUT.

The FPGA itself is Flash-based, as mentioned previously, because this technology is highly resistant to gate configuration errors.

Additionally, the components on the FPGA chip responsible for controlling and monitoring the SUT should not experience failure under radiation exposure. To mitigate this risk, we implemented the well-established countermeasure Triple Modular Redundancy (TMR). We integrated TMR into all Verilog modules that interact with the neuromorphic architecture, thereby protecting all communications with the SUT against radiation. This ensures that no errors occur within the communication layers around the SUT, allowing us to rely confidently on the output data that reflect the internal data of the SUT. In Figure 4.3 the modules protected by TMR are colored green.

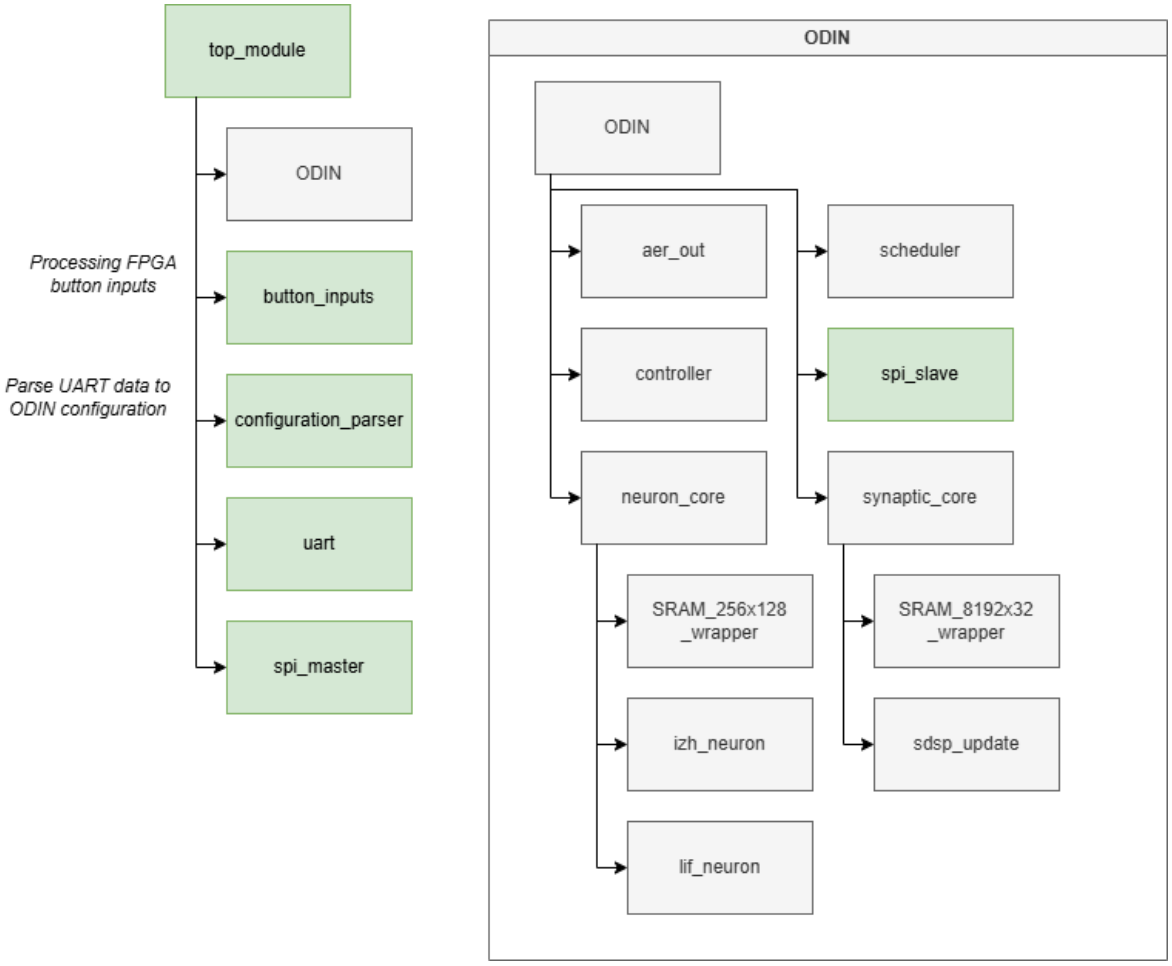


Figure 4.3: The modules and their TMR protection (green is protected)

Radiation experiments The process works as follows: on the side of the radiation room is a door, when the door opens the radiation beam enters the room. We placed the FPGA in front of the beam, we aligned in such a way that the beam hits the programmable part of the board and we kept the other peripherals of the board as much as possible out of the beam line because we do not want the device to break or fail on something else than the SUT, which is the neuromorphic architecture; see Figure 4.4.

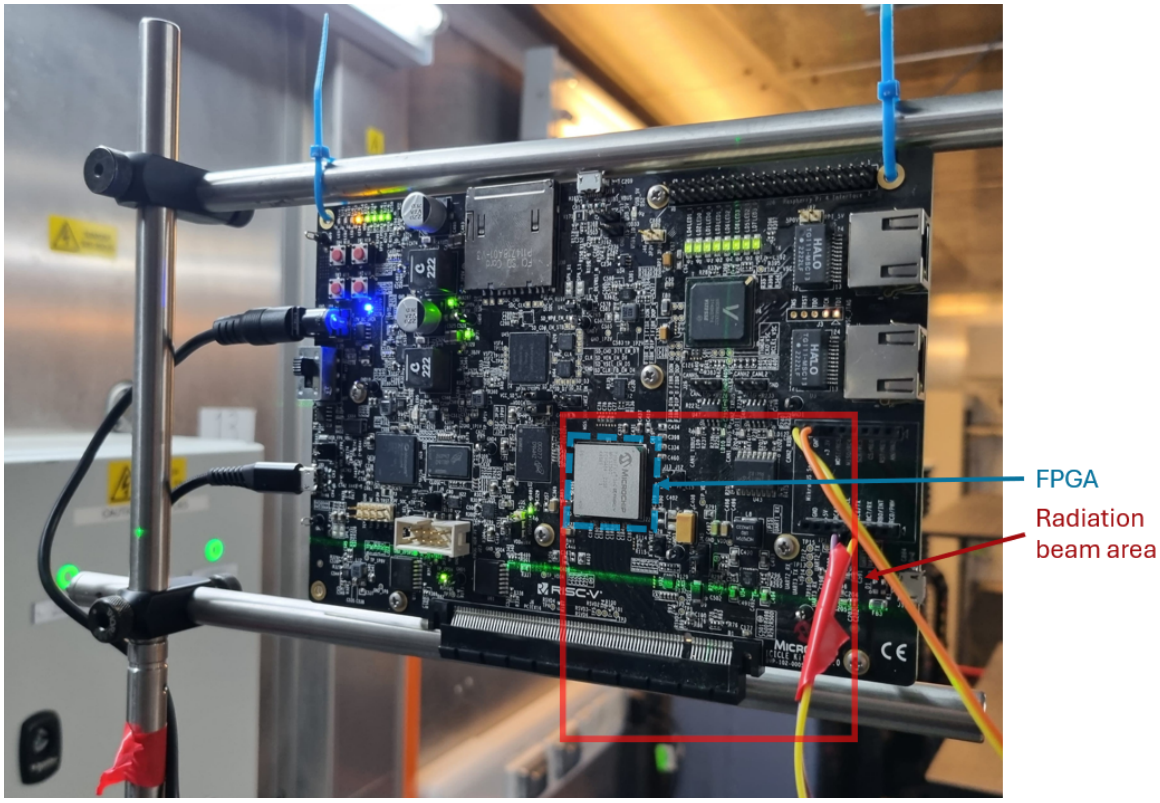


Figure 4.4: The device under test

Communication To collect results while radiation hits the device we setup an UART over Ethernet connection to the device to monitor and control the device. The monitoring is used to detect the effects of the radiation; this is done in two different ways. The key metric is the model accuracy, the assumption is that radiation will distort the model weights, the changed weights will change the accuracy, so we can see the impact of the radiation accumulating over time. The other metric is just reading all the memory of the SUT, comparing the memory state to the initial memory state provides insight into memory changes. An important side note of this memory comparison approach is that in the case of online learning the weights are changed by the model as well, so it is harder to tell what radiation did in that case.

Time window utilization For radiation testing, we had a time window of 48 hours. This fixed time window requires optimal planning for the test scenarios. In preparation for radiation testing, several test cases are prepared. We want to know what happens when radiation affects SUT when online learning is disabled, and also we want to know what happens when we enable online learning. To study the effects of a longer execution time, we distinguish between a short run and a longer test run. The short run consists of running digit recognition on 6000 images, and the larger test run uses 60000 images.

Test execution A test run is defined by a file; this file contains all the commands which should be executed. We use the csv [6] format for this. The file will be called the run configuration file or test configuration file later in the report. The first started with configuring the ODIN architecture, configuring the neuron memory, and then loading the initial weights into the synapses. The system is then ready to process the commands, which are also listed in the test configuration file. At the end, the file should put commands to readout the neurons and synapse memory to be able to detect the radiation effects on the memory.

4.5 Radiation simulation

To obtain additional insights, we can perform a simulation in which we manually inject faults into the system. To get a meaningful simulation we have to collect data from the beamline and measure the bitflip rates. The beamline we used is the ChipIr at the Appleton Labratory [7]. If we know the chance of a bitflip occurring, we can use that chance within the simulation as well.

FPGA simulation For the simulation, we use the FPGA as target, because we have write access to the synapse memory, see section 4.2, we can easily inject a bitflip there.

The process of injecting a fault will be as follows: First, we need the rate at which bitflips occur. With this rate, we need to create a run configuration which does the interference, but in between sending the images, it should send the command to modify the synapse memory to inject a bitflip.

The process of injecting a bitflip in a synapse will then be as follows: first, a random number is generated to select one of the 256 synapses, and the selected synapse will be read from the FPGA. Another random number is generated to select

[6] csv: comma separated values
[7] See https://www.isis.stfc.ac.uk/Pages/Chipir.aspx

a bit within the 1024 synapse bits, the selected bit will be inverted. Then the changed synapse data will be written back to the FPGA. For the random numbers, we use a pseudo random function; this allows us to get the same random numbers when we apply the same seed; this is useful because we want to have 2 runs with identical bitflips, one run without learning and one run with learning to be able to compare them. So every seed is used twice to generate a test configuration for both learning and non-learing. To generate the next test configuration set, a new seed, which was not used before, should be chosen.

For executing the interference, we have a run configuration file including all the UART commands, in this file the synapse commands, which do the fault injection, should be added. Subsequently, the configuration file can be executed in a conventional manner and the results can be collected in the same way as used for the other tests without simulated faults.

Chapter 5

Results

We categorize the results into three distinct groups. We have the baseline results, obtained without any radiation. The second category contains results derived from real radiation exposure, utilizing a neutron beam in the United Kingdom at the ChipIr facility. The final category consists of data acquired by manual fault injection, with fault rates calibrated according to the measurements taken at the radiation beam.

To retrieve results for the first two categories, data acquisition was performed via the FPGA, employing the UART communication between the FPGA and the host system. This facilitated a comprehensive analysis enabled by access to the entire memory of the neuromorphic architecture. Image interference serves as a pivotal factor in obtaining results; in addition, memory dumps were collected for detailed analysis within this chapter. In each test, two configurations were executed. The first configuration excludes learning, thus only doing interference. In contrast, the second configuration incorporates online Spike-Dependent Synaptic Plasticity (SDSP) learning, allowing it not only to interfere but also to adapt and adjust in response to disturbances. For the last category, we injected faults into the FPGA to obtain further insights into the interaction of learning mechanisms with radiation.

The result data are organized as follows: initially, we examine the accuracy for each image class individually. This is followed by an analysis of the temporal changes in accuracy, providing insights into the accumulation of radiation effects over time. Subsequently, a fault analysis is conducted wherein the fault probability is calculated. We then present the findings from the radiation simulation, which are derived from the measured fault probability at the radiation beam. The concluding section of the results will explore the spatial utilization of online learning implementations on FPGAs and will draw a comparison with the spatial utilization characteristic of the well-established radiation protection technique known as Triple Modular Redundancy (TMR).

5.1 Class accuracy

Accuracy metrics are presented in the subsequent subchapters. Throughout each radiation test and simulation run, the accuracy of interference was assessed and recorded on a per-image basis. This approach allows for the graphical representation of classification accuracy for each class, thereby enabling a more nuanced analysis of classification performance.

For an overview of the results, see Table 5.1. From this table we can see the following; first we have a stable accuracy above 80% as long as there is no radiation. The interference runs without learning are deterministic, which makes it run without any deviation. Not every test was executed equally often; this is due to the execution time of the tests, larger tests consume much more time because of all the UART overhead.

The 4 tests configurations without radiation show the stability of the neuromorphic architecture, the accuracy is consistent over multiple runs, for learning, there is an accuracy deviation between the runs, but overall the accuracy is not changing much.

The next 3 tests configurations with radiation show that for the shorter radiation period the accuracy is influenced, both lower and higher accuracies are measured for learning and non-learning. The interesting thing is that on average both learning and non-learning accuracy increases, although it is by a very small margin of 0.03%. For the longer test run with the 60000 images we see a larger deviation, and the average accuracy also drops down. The drop in accuracy is mostly caused by two runs which very quickly showed a big drop in accuracy; for those it is probability caused by faults in different parts of the system, because when doing simulation we do never see such a quick accuracy drop after such a short time (at a relatively low number of bitflips).

The two runs of 4 hours of simulation with the 60000 images show that after a longer time the learning starts to be a little bit more accurate than running without learning. The deviation in accuracy is higher for the learning configuration, indicating that the learning is overreacting sometime, nevertheless, the learning still outperforms the non-learning.

To get to a low accuracy much more radiation time is needed, so we tested for an extended period of 840 hours (35 days) to get down to a lower average accuracy, with those runs we also see instances for which the non-learning accuracy drops to +/-10%, which is equal to assigning the classifications at random, meaning that the knowledge (configured weights) about the digits is gone, and the system no longer is capable of meaningful classification. On average, non-learning is still at 24.95% accuracy, and delta with learning has grown to +/-5%, because learning is

on average 29.86% accurate. The maximum deviation between runs for learning did not grow much larger than for the shorter simulation test of 4 hours (10.21 vs 10.85). In contrast, the deviation for the non-learning runs has grown and exceeds the deviation for learning; the inability of responding to disruptions caused by the simulation causes the runs without learning to have a higher maximum deviation (11.63).

So, based on the class accuracy results (as shown in Table 5.1), we see positive effects of learning, those effects only become visible after a longer radiation time, because at shorter radiation time the learning tends to stabilize with a slightly lower accuracy than the pre-trained weights we put into the system.

Learning	Radiation	Images	Accuracy (%)	Deviation	Iterations
no	no	6000	82.92	0.00	100
yes	no	6000	82.26	2.89	100
no	no	60000	82.38	0.00	45
yes	no	60000	81.37	2.32	45
no	yes (+/-20 minutes)	6000	82.95	2.35	18
yes	yes (+/-20 minutes)	6000	82.33	1.54	18
no	yes	60000	-	-	0
yes	yes (+/-4 hours)	60000	75.81	30.87	6
no	simulation 4 hours	60000	81.29	5.16	8
yes	simulation 4 hours	60000	81.75	10.21	8
no	simulation 840 hours	10000	24.95	11.63	45
yes	simulation 840 hours	10000	29.86	10.85	45

Table 5.1: Accuracy and deviation across different configurations

5.1.1 Without radiation

Using 6000 images For this test, the first 6000 images of the MNIST dataset are used. Without learning, the ODIN architecture behaves deterministic, yielding the same results for every iteration. At the left side of Figure 5.1 this is visible in the absence of error bars. The presentation of the first 6000 images of the MNIST dataset to the ODIN architecture when learning is enabled influences the weights of the synapses. The learning is non-deterministic and is driven by the behavior of the spikes. This results in different accuracy results when executing multiple runs with the exact same 6000 images in the exact same order. On the right side of Figure 5.1 the average results of 100 runs are shown. The maximum deviation is +/-2.89%.

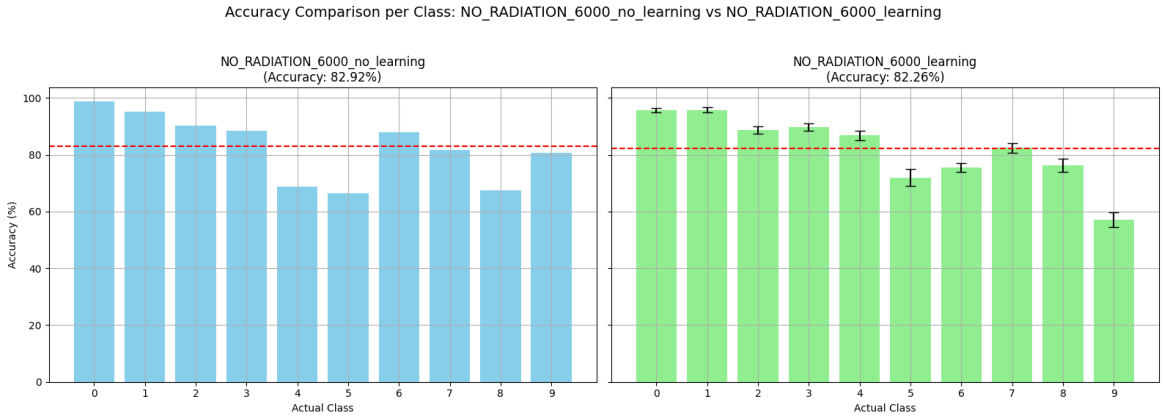


Figure 5.1: Accuracy per class on 6000 images without radiation
Left: no learning, Right: with learning

Using 60000 images When using all 60000 training images of the MNIST dataset, the accuracy is a bit lower compared to the first 6000 images, this is due to the dataset distribution, the first 6000 images we used are easier to classify with our pre-trained weights, than the 54000 images following those. On the left side of Figure 5.2 the class accuracy without learning is shown, this is higher than the accuracy with learning. When running on all 60000 training images with learning, we see deviation between runs., this is shown on the right side of Figure 5.2. The maximum deviation is +/-2.32%.

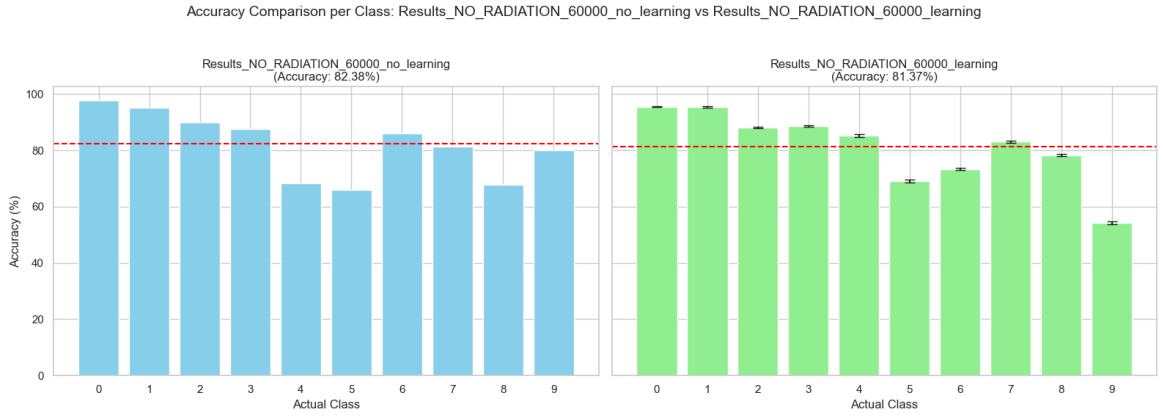


Figure 5.2: Accuracy per class on 60000 images without radiation
Left: no learning, Right: with learning

5.1.2 With radiation

Using 6000 images When radiation is applied in the same configuration with the same 6000 images, we see deviations in the accuracy. The left side of Figure 5.3 shows the results of 18 runs, each run taking approximately 22 minutes. The maximum deviation is +/-2.35%. When radiation is applied, similar deviations in accuracy as before between differences runs are visible. On the right side of Figure 5.3 the results of 18 runs are combined, each run taking approximately 21 minutes. The maximum deviation is +/-1.54%.

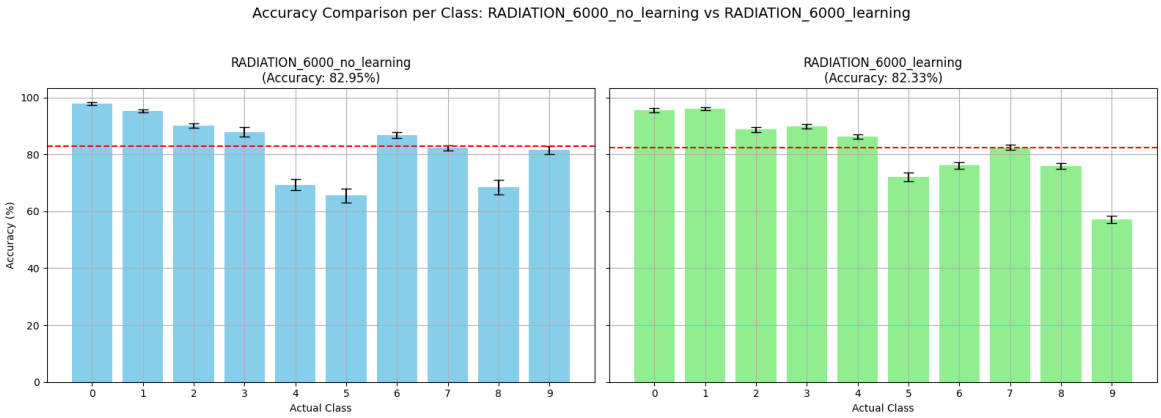


Figure 5.3: Accuracy per class on 6000 images with radiation
Left: no learning, Right: with learning

Using 60000 images Then, under radiation, the same 60000 images are processed multiple times by the system. There are 2 runs out of the 6 that showed a large degradation in performance, which causes the accuracy to be lower for those 60000 images. Each of those 6 runs took approximately 4 hours to complete. The averages of those runs are shown in Figure 5.4. The maximum deviation is +/-30.87%.

For the test runs with 60000 images with radiation we do not have data on non-learning because of the limited testing time, and because non-learning behaves deterministically so we can test and compare that easily later on by doing some simulations, see subsection 5.1.3.

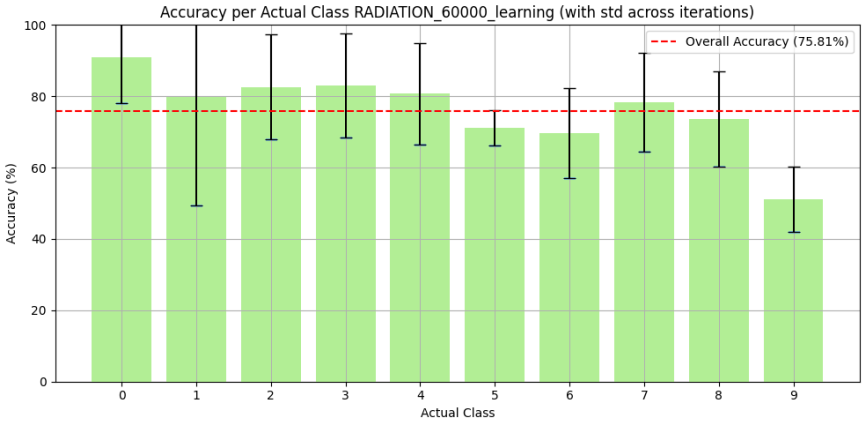


Figure 5.4: Accuracy per class on 60000 images with radiation*with learning*

5.1.3 With simulation

The need for simulation comes from the missing tests with 60000 images under the beam. Because of time constraints, we did not run tests with all 60000 images without learning enabled. To be able to compare the learning effects on those 60000 images, we ran a simulation. For this simulation, we used the measured fault rate from section 5.3 and used the calculated $MTTF_{bitflip}$ for the injection of faults.

To obtain higher accuracy results in the simulation, we changed the learning parameters as shown in Table 5.2. This gives an increase in accuracy for the learning part. The chosen parameters are chosen by a trial-and-error process. In chapter 8 further research directions on learning parameters will be explored.

Parameter	Radiation	Simulation	Max
`Threshold membrane learning`	31	239	255
`Calcium variable threshold 1 learning`	2	2	7
`Calcium variable threshold 2 learning`	3	4	7
`Calcium variable threshold 3 learning`	4	6	7

Table 5.2: Parameter values for radiation, simulation, and maximum settings.

Using 60000 images For running the larger set of 60000 images without learning, we do not have data collected from the radiation experiments; therefore, we ran simulations for which we applied bit flips with the same rate [1] to the memory as those occurring during the radiation experiments. The accuracy decreases when radiation simulation affects the device; in Figure 5.5 we see the averages of 8 simulation runs.

[1] The fault probability is calculated in section 5.3

The left side of Figure 5.5 shows the averages of the runs without learning, and the maximum deviation is +/-5.16%. The right side of Figure 5.5 shows the averages of the runs with learning, where we have a maximum deviation of +/-10.21% in the accuracy between runs.

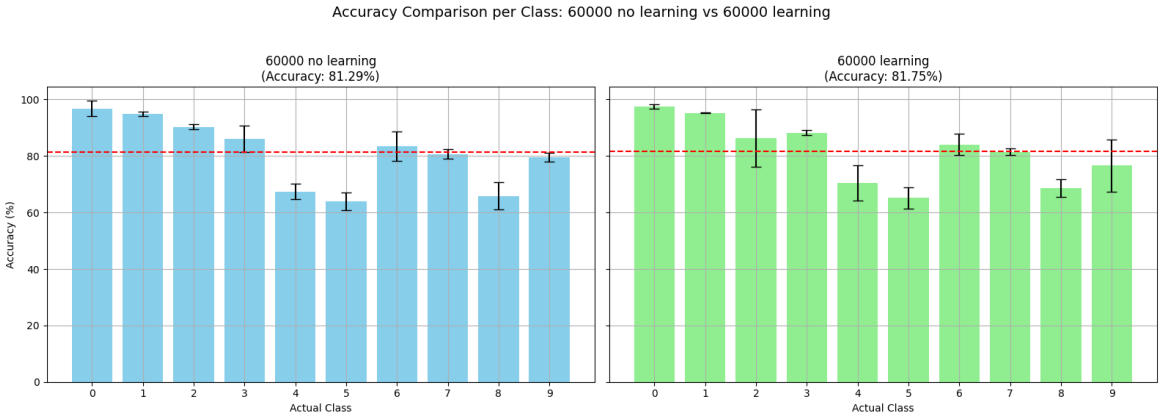


Figure 5.5: Accuracy per class on 60000 images with simulated radiation
Left: no learning, Right: with learning

5.1.4 With simulation periods

To gain more insight into how the neuromorphic system will behave when exposed to a longer period of radiation, we extended the simulation duration. We still use the same learning parameters and the same fault rate as given in subsection 5.1.3.

For these longer simulation runs, we use periods that reflect an equivalent of the radiation time under the beam at the ChipIr testing facility. In each period, we inject an equivalent amount of bitflips as the ChipIr beam would have caused. In each period, we do one interference run for all the 10000 test images. At the end of each period, we measure the accuracy and process this data to gain insight into the resistance to radiation over time.

Periods of 20 hours After running 14 periods of 20 hours the accuracy has dropped a bit. It started at 82.18% for non learning and at 81.04% for learning. The 14 periods count up to a total of 280 hours of radiation simulated. The result of the simulated radiation is shown in Figure 5.6

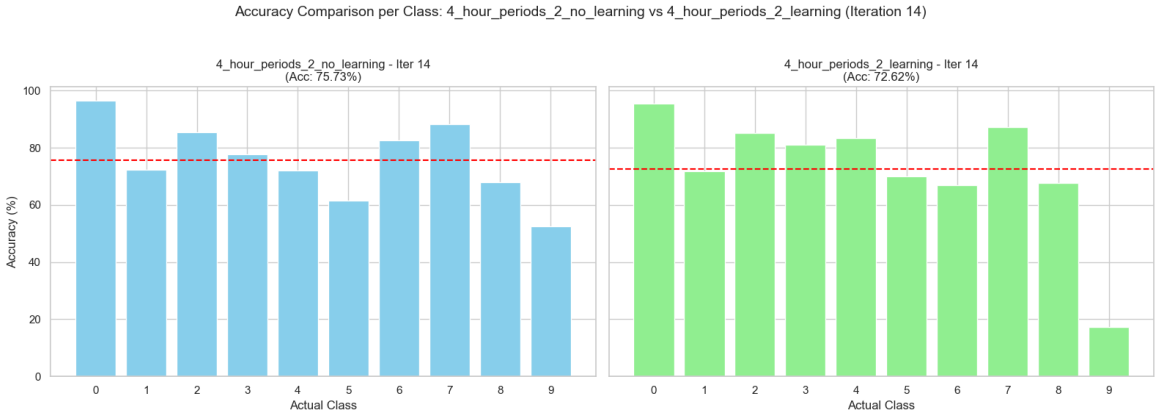


Figure 5.6: Accuracy per class on 10000 images with simulated radiation after 14 periods of 20 hours
Left: no learning, Right: with learning

Periods of 40 hours After running nine periods of radiation intervals of 40 hours, we also see a degradation of accuracy. This is at 360 hours of radiation, while the previous plot Figure 5.6 was 280 hours of radiation. The longer time caused a further drop in accuracy, as shown in Figure 5.7.

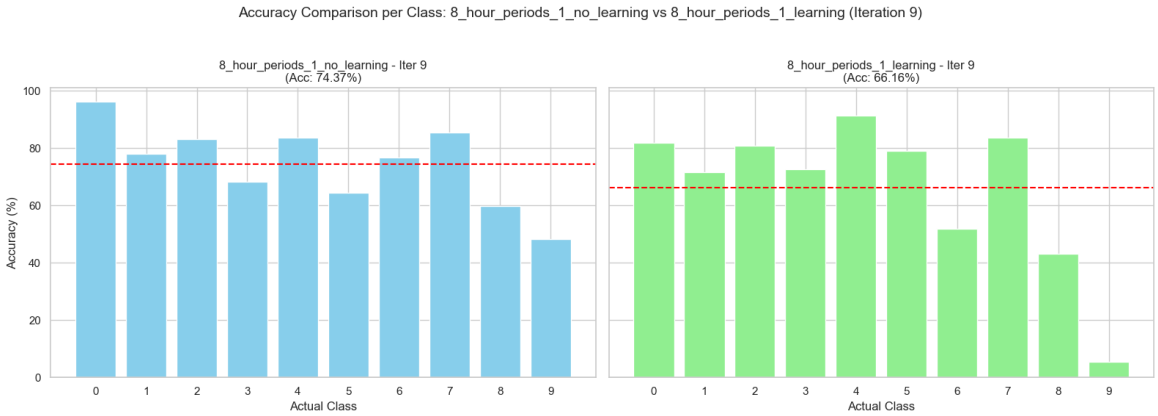


Figure 5.7: Accuracy per class on 10000 images with simulated radiation after 27 periods of 40 hours
Left: no learning, Right: with learning

Periods of 120 hours For the next plots, we used a longer period of radiation between interference runs. We use 8 periods, of which the first period is without radiation, so effectively we use 7 periods of 120 hours of simulated radiation equivalent to the measured bitflip rate at the ChipIr beam. The plot in Figure 5.8 shows

that, in the absence of learning, the accuracy is retained only for the digit '2', with negligible accuracy for other digits. However, with the implementation of learning, digits '1', '2', '3', and '4' are classified with reasonable precision. A subsequent experiment, conducted with a different random seed as shown in Figure 5.9, resulted in all images being classified as digit '6' without the influence of learning, resulting in an accuracy of 0% for all other digits. In contrast, when learning is applied, digits '3', '6', and '7' continue to exhibit reasonable accuracy.

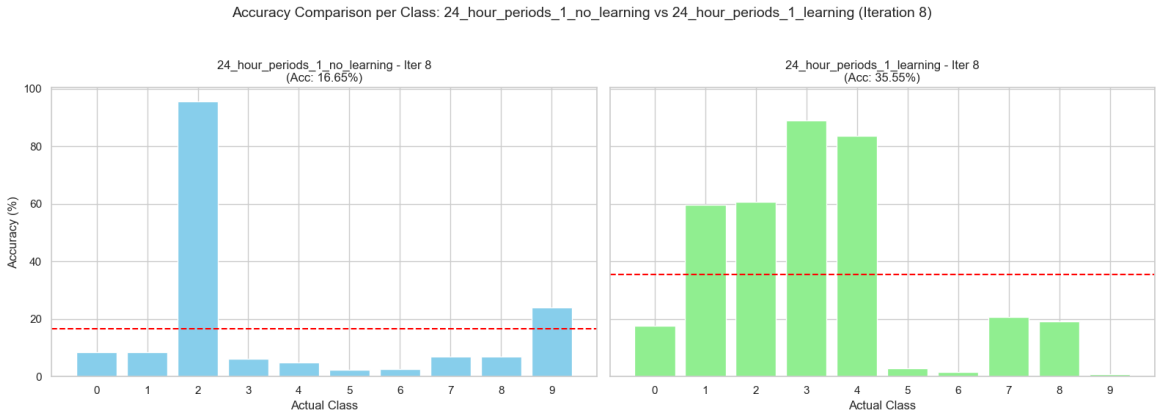


Figure 5.8: Accuracy per class on 10000 images with simulated radiation after 7 periods of 120 hours
Left: no learning, Right: with learning

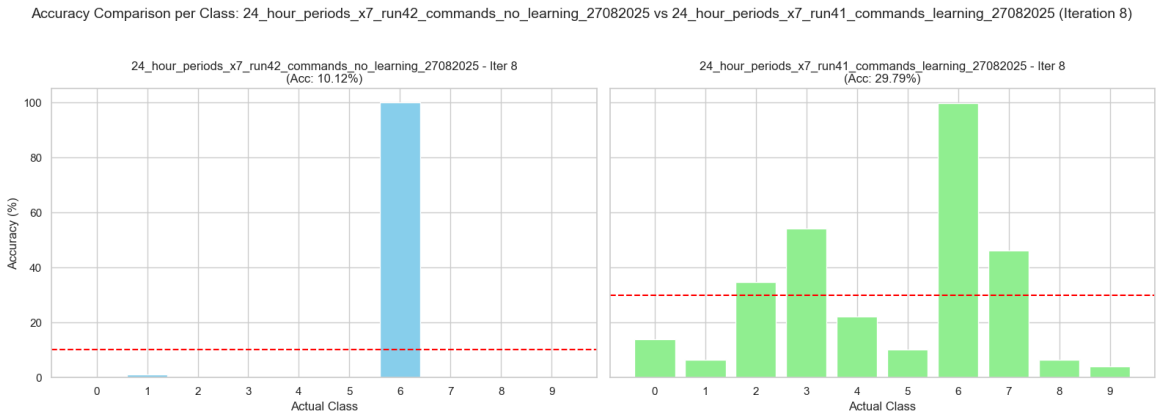


Figure 5.9: Accuracy per class on 10000 images with simulated radiation after 7 periods of 120 hours
Left: no learning, Right: with learning

Periods of 120 hours average To obtain a more comprehensive understanding of the average accuracy behavior over an extended duration, we conducted 45 tests in which accuracy was assessed after 840 hours of simulated radiation exposure.

For the resulting average accuracy over all the simulation runs and the lowest and highest accuracy after 840 hours of radiation, see Table 5.3. The data shown in that table is also plotted to make the standard deviation visible; see Figure 5.10. From this table, we can see that the minimum accuracy with learning is always higher than the minimum accuracy without learning. This minimum value is selected from the accuracy value at the end of each run, so it is the overall lowest value of all runs. For the mean we see a partial overlap of the standard deviation between the learning and non-learning, but important here is that both the lower-bound and upper-bound show a higher value than the std from the non-learning mean.

Metric	No Learning (%)	Learning (%)
Lowest accuracy	10.12	14.91
Highest accuracy	50.66	52.61
Average accuracy	24.95	29.86

Table 5.3: Accuracy statistics with simulated radiation after 840 hours

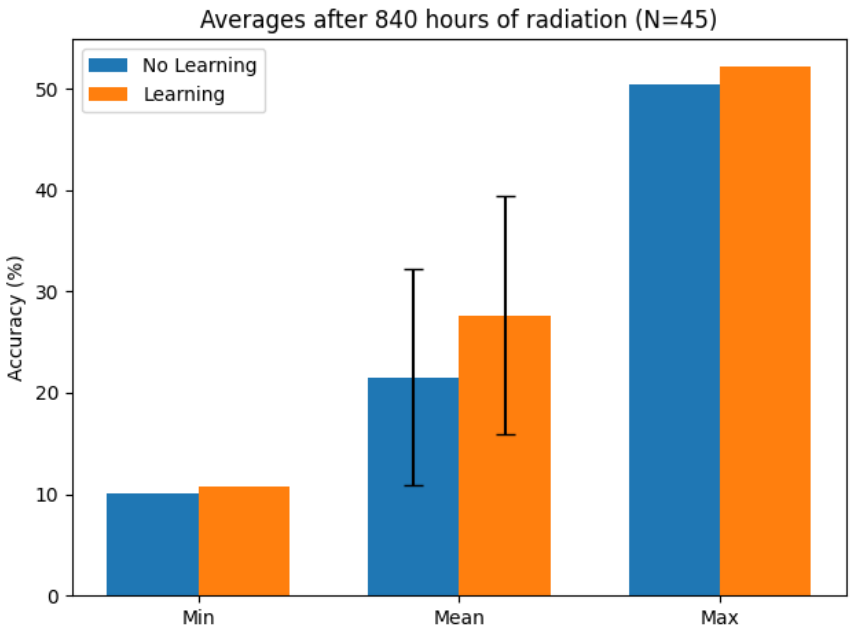


Figure 5.10: Min, mean and max accuracy on 10000 images with simulated radiation after 840 hours

More detailed results of these experiments are presented in Figure 5.11, in this the results per class are shown. The digits '0', '2', and '7' are the best scoring digits when learning is enabled, all digits, except digits '6' and '9', are on average more accurately classified by learning than by non-learning. Also interesting to see is that without learning all digits get an accuracy on some run (see the std markers), but with learning 5 digits do never get an accuracy of 0%.

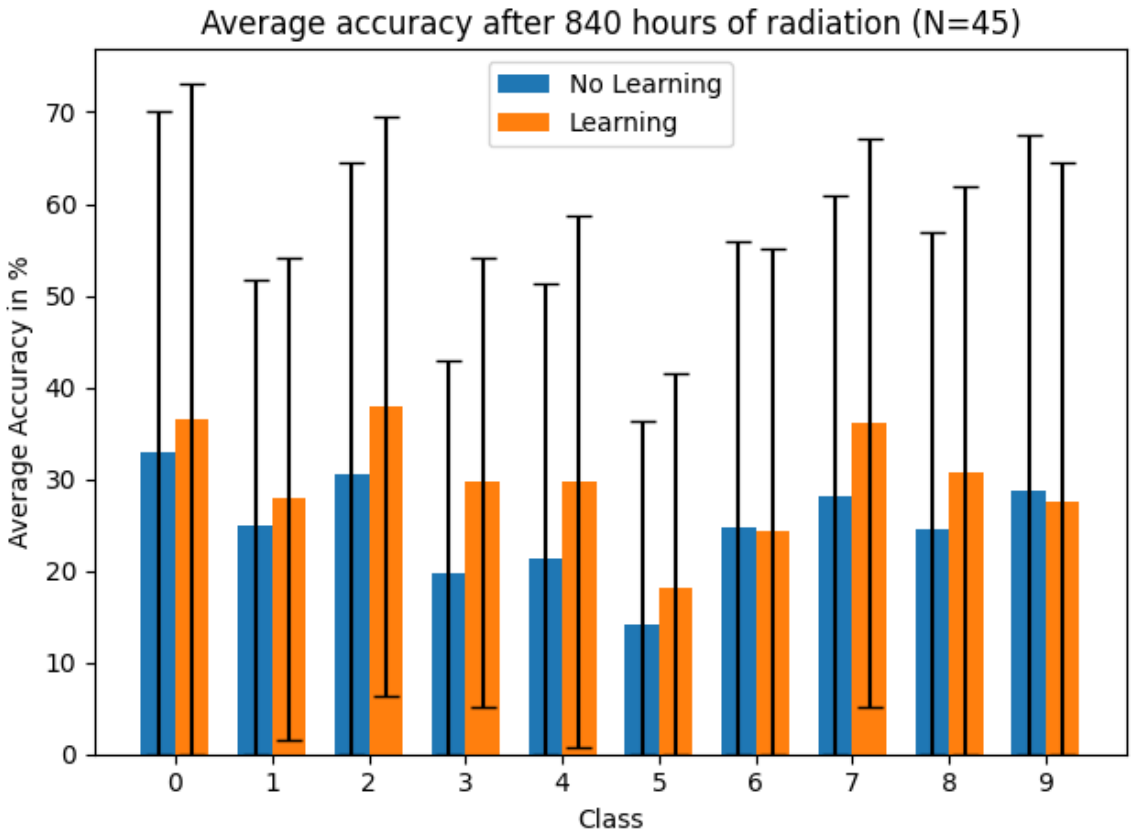


Figure 5.11: Average accuracy per class on 10000 images with simulated radiation after 7 periods of 120 hours

5.2 Accuracy over time

A very interesting way of looking at the results is by calculating the average accuracy over time. This gives insight into how the longer execution time, and thus the higher radiation exposure, has an impact on the accuracy. For this we first look at how the system behaves without radiation, we see that the 6000 and 60000 images show consistent behavior without learning. With learning enabled, we see deviations between the runs because of the non-deterministic behavior of the learning, but the results are still showing stable learning accuracy over multiple runs.

As soon as we start applying radiation we see deviation for both learning and non-learning accuracy. For the 6000 images the deviation between runs is small. For the 60000 images we see mostly small deviations as well, but there are also two instances where the accuracy drops very quickly down. This drop is unexpected and could have been caused by failures in other parts of the system.

To get more insights we ran simulations with the 60000 images, with those simulations we never saw an accuracy drop as quickly as under the beam tests. With this simulation we are able to compare the learning with the non-learning tests. Overall we see a stable accuracy around 82% for both learning and non-learning. However, in one case we saw the accuracy drop 7% without learning, while the learning only allowed a 3% drop (both learning and non-learning got exactly the same bitflips here), so here we see a positive effect of learning on the accuracy.

Longer simulation runs have been done as well, for that periods of radiation have been simulated, period lengths ranging from 20 to 120 hours of simulated radiation are applied to the ODIN memory. For shorter runs with smaller periods, we see a gentle accuracy drop for both learning and non-learning, the non-learning accuracy is a bit higher for those runs. When we expand to a longer duration with larger periods of radiation simulation we see two things happening. First, we see the learning overreacting after the first 120 hours of radiation simulation, meaning that the accuracy of learning drops quickly down. The second observation is that after some time the non-learning accuracy starts to drop down as well, the non-learning accuracy continues to decline, sometimes even all the way down to 10% accuracy while the learning accuracy never drops down that far, meaning that at some point the learning accuracy is higher than the non-learning accuracy. Overall after 7 periods we see the learning averages at a 5% higher accuracy than the non-learning.

Looking at averages of learning and non-learning accuracy tells us that without learning it is possible to loose all accuracy after 480 hours of simulated radiation, while with learning the system always keeps working, albeit that the accuracy drops down as well. Getting below 75% and 50% accuracy on average happens for both

learning and non learning equal numbers of intervals, going below 25% accuracy on average does not happen for learning, but happens after 840 hours for non-learning.

So, as shown in the accuracy over time plots, learning has a positive effect on accuracy after a longer period of radiation exposure. On shorter runs the learning does not show positive effects and suffers from over-reacting on synapse weight changes.

Side note *On the accuracy over time graphs we see a drop in accuracy occurring on all tests we did. This drop is around the 1500 images. This is explained by the unbalances in the MNIST dataset and our relatively large differences in accuracy per digit. As we saw in section 5.1, not every digit is classified with the same accuracy; see Appendix D for more information.*

5.2.1 Without radiation

To get a baseline and a reference point we first show the results for learning vs non learning without any radiation influences on the system.

Using 6000 images When doing interference on 6000 images without learning and without radiation the results are always the same, thus it looks like only one line in the upper part of Figure 5.12. When learning is enabled a more diverse non-deterministic accuracy will follow, so the lines are all a little bit different in the lower half of Figure 5.12.

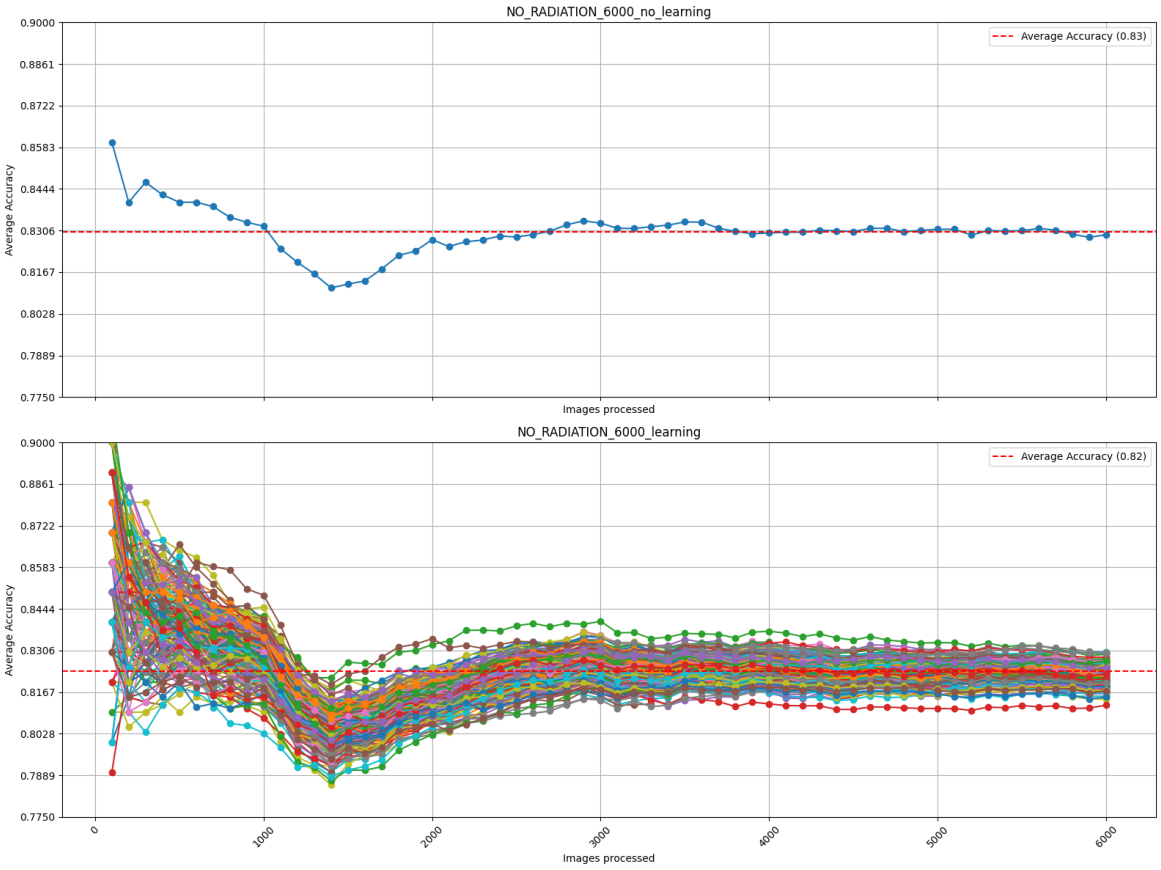


Figure 5.12: Accuracy over time for 6000 images without radiation
Top: no learning, bottom: with learning

Using 60000 images For 60000 images the Figure 5.13 learning scores somewhat lower than the non-learning part, but apart from that small offset in accuracy there is not a real difference visible.

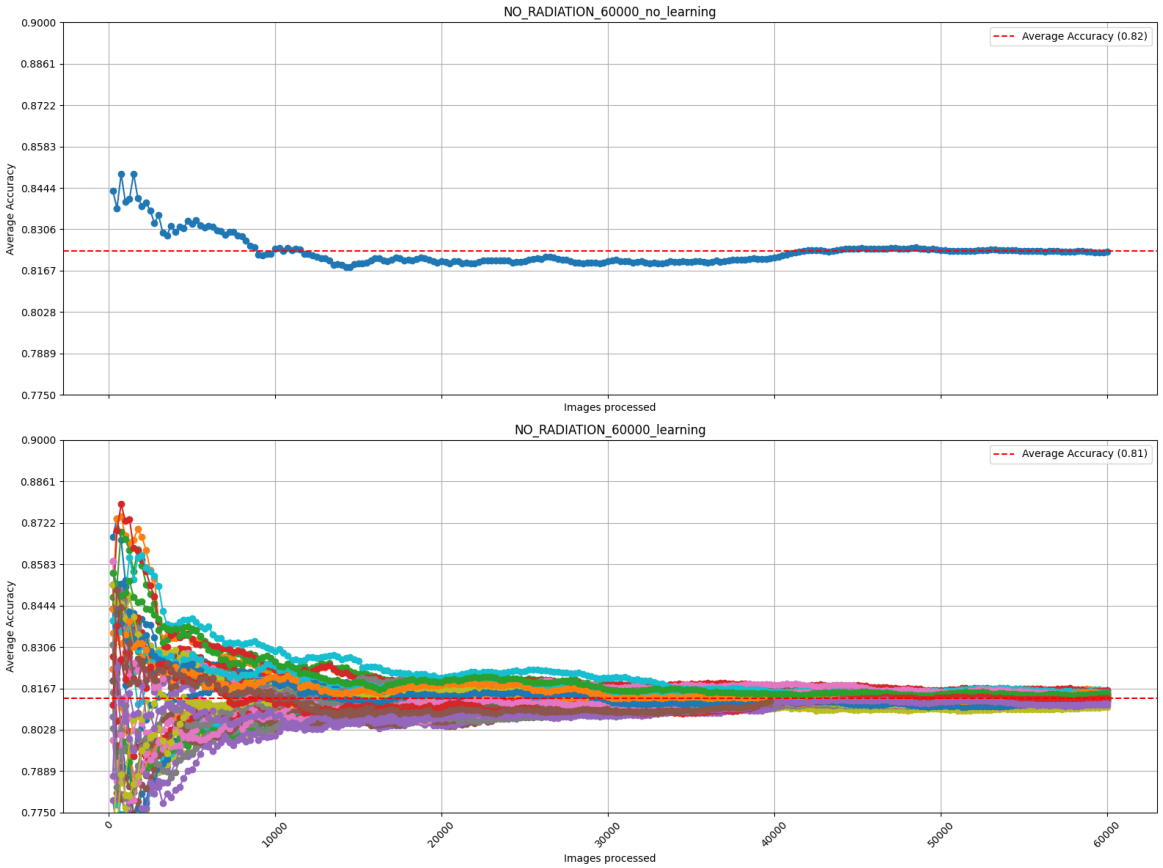


Figure 5.13: Accuracy over time for 60000 images without radiation
Top: no learning, bottom: with learning

5.2.2 With radiation

Measurement of the accuracy again with radiation that affects the system will provide additional insights.

Using 6000 images For 6000 images, we see very similar results for both learning and non-learning in Figure 5.14. Still, the learning scores are a little bit lower, but the offset between learning and non-learning is not really changed.

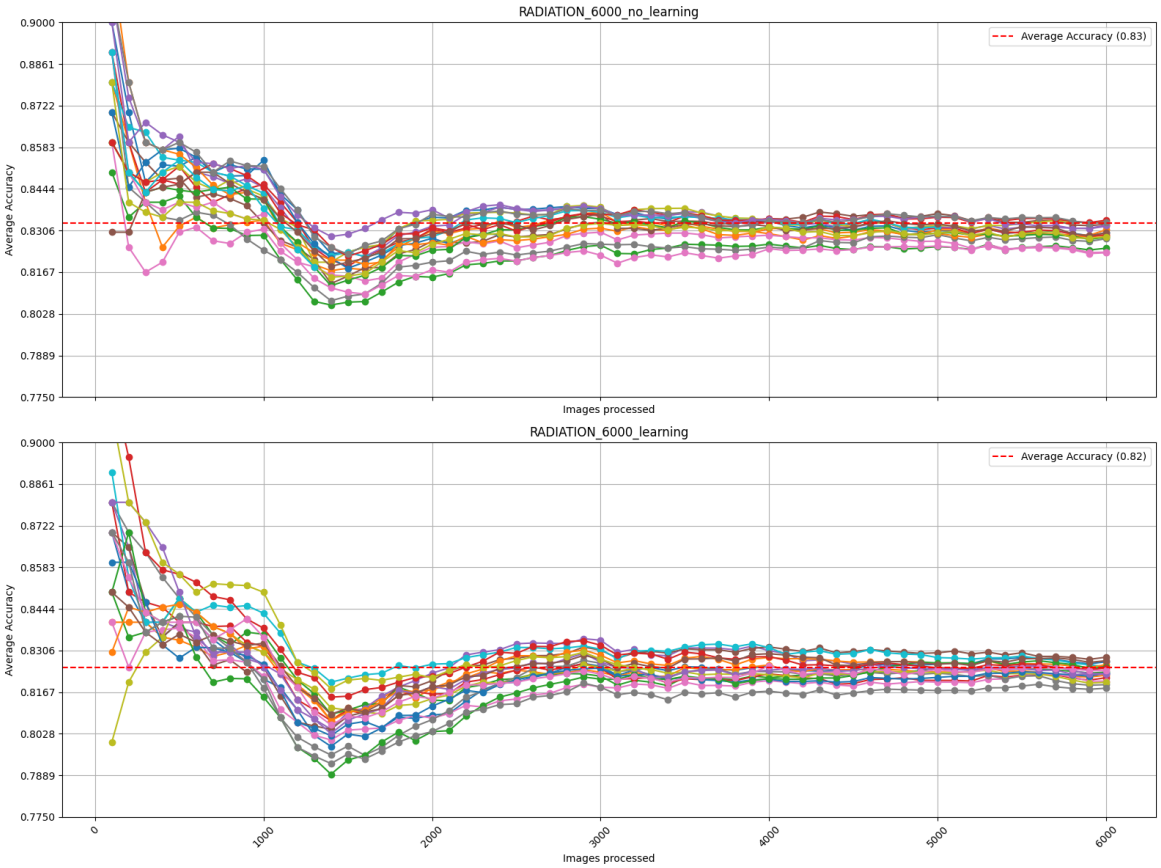


Figure 5.14: Accuracy over time for 6000 images with radiation
Top: no learning, bottom: with learning

Using 60000 images For this test we only have data from the test with online learning enabled as shown in Figure 5.15, this makes comparisons impossible. Because of this, we also performed some simulations as described in subsection 5.2.3.

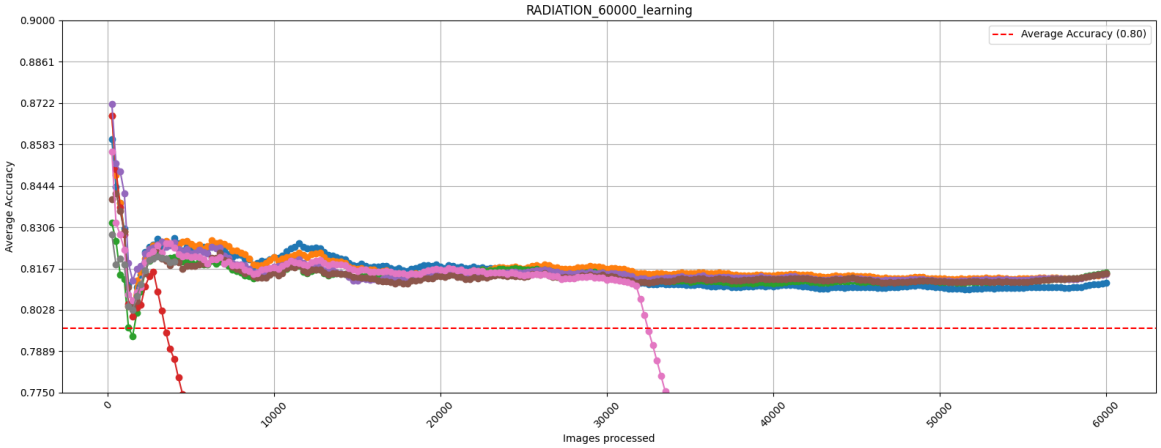


Figure 5.15: Accuracy over time for 60000 images with radiation*with learning*

5.2.3 With simulation

To make comparison to study the effects of online learning easier to see, we developed a fault injection mechanism. This fault injection uses a pseudo-random generator so we have the exact same bitflips happen for both the learning and non-learning part.

Using 60000 images In Figure 5.16 the results are shown.

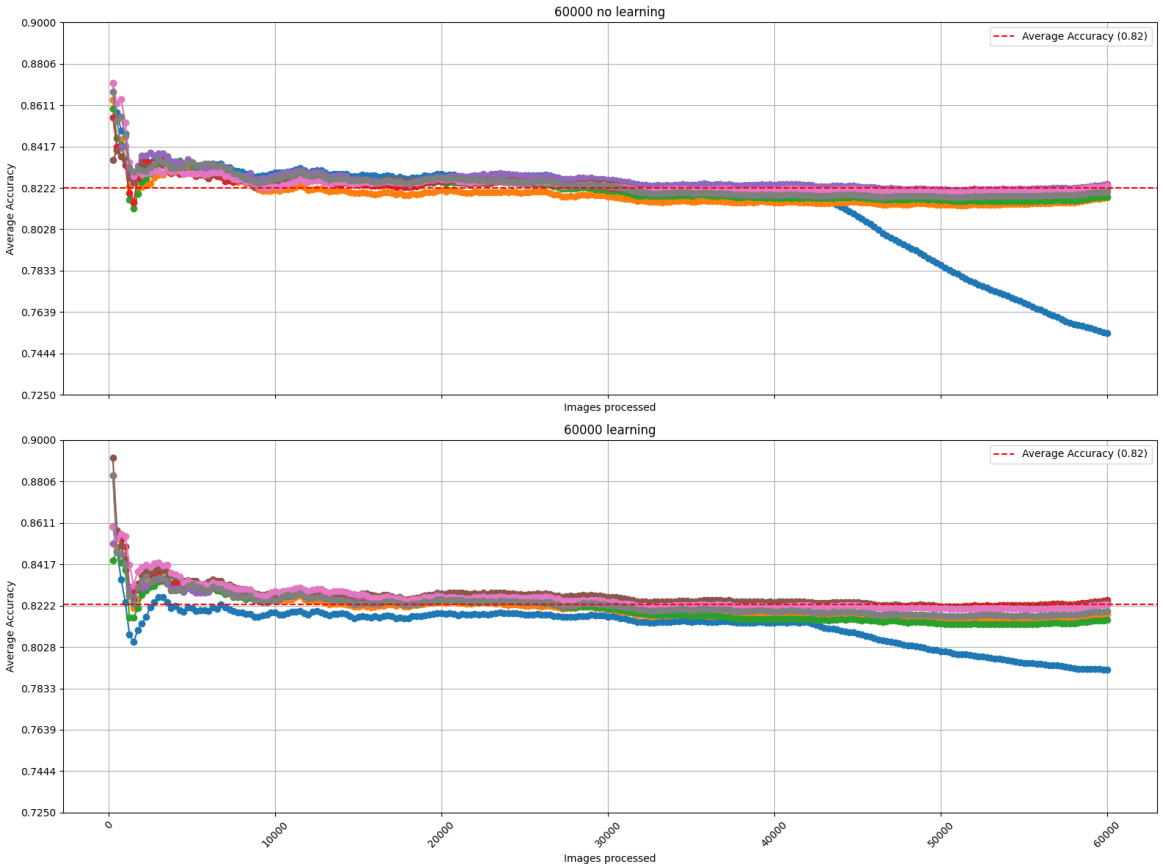


Figure 5.16: Accuracy over time for 60000 images with simulated radiation
Top: no learning, bottom: with learning

5.2.4 With simulation periods

After we run the longer simulations, the results show variability. In shorter simulations, the learning mechanism exhibits lower accuracy compared to the non-learning configuration. However, with prolonged simulations, the learning algorithm demonstrates superior accuracy relative to its non-learning counterpart. In our plots, time is denoted as periods, where each period corresponds to the duration of beam exposure. For example, a period of 20 hours signifies exposition to the beam at ChipIr for

20 hours. During each period, interference is executed once, which implies that in longer periods, there is an increased simulated waiting time between interferences. The period 0 in the plots shows the baseline, in this period only interference was executed without the application of any simulated radiation.

Periods of 20 hours We start with periods of 20 hours. The accuracy of learning vs without learning are close to each other as shown in Figure 5.17.

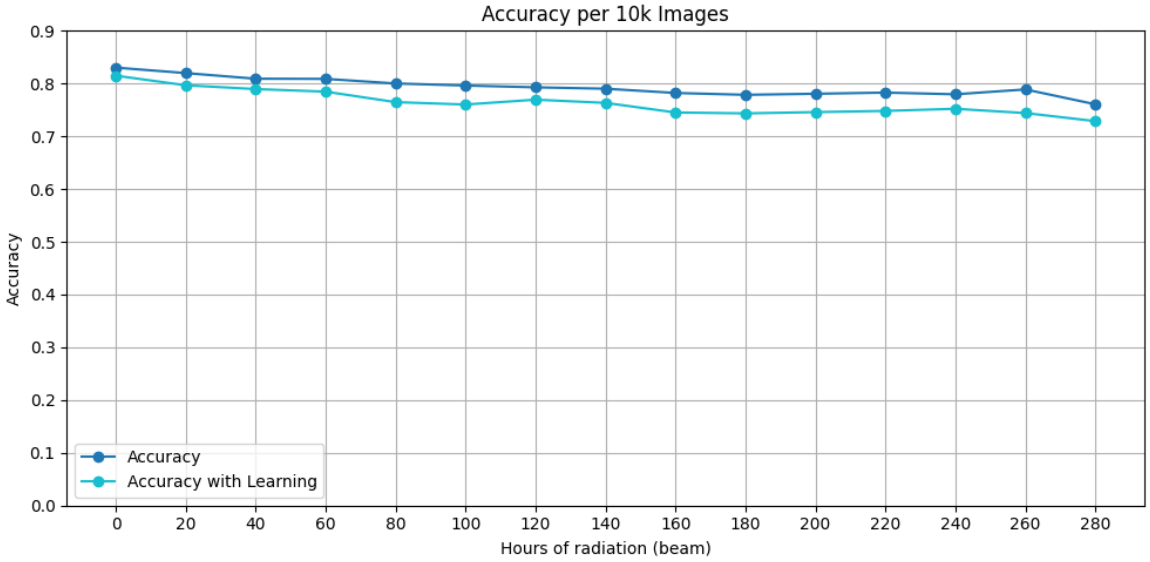


Figure 5.17: Accuracy over time for 10000 images with simulated radiation

Periods of 40 hours With longer periods of 40 hours we see similar results for the accuracy of learning vs. without learning in Figure 5.18. Due to longer periods of radiation, the total execution time has increased. Because of this additional time, we see the accuracy dropping further down.

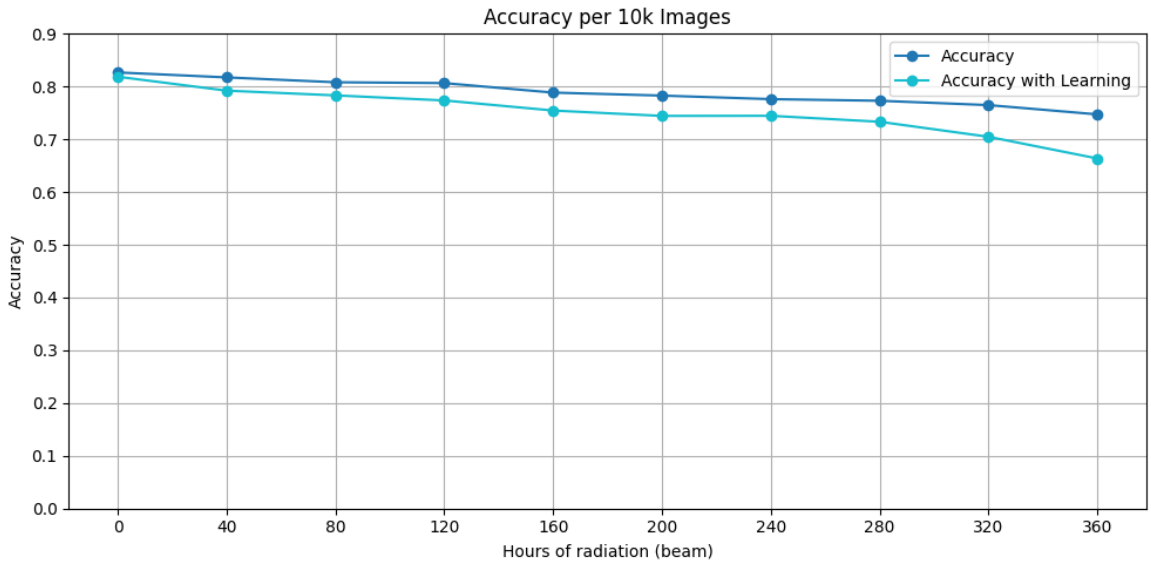


Figure 5.18: Accuracy over time for 10000 images with simulated radiation

Periods of 120 hours When we expand the period further, we choose periods of 120 hours. With this duration, we see the accuracy without learning dropping to approximately 0.1 after 5 iterations in Figure 5.19.

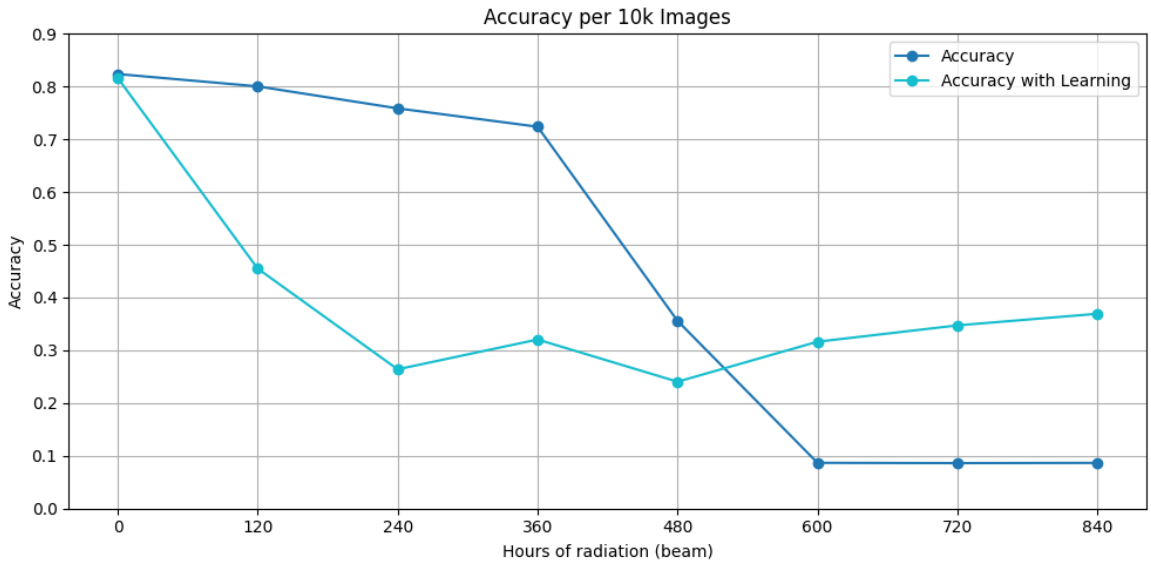


Figure 5.19: Accuracy over time for 10000 images with simulated radiation

In the next run, shown in Figure 5.20 the accuracy without learning drops to 0.17. Both the learning and non-learning configurations rapidly experience a decline in their accuracy. However, at the second iteration, the learning accuracy becomes higher than the non-learning accuracy. This margin is maintained and expanded

over time. The bitflips introduced between 240 and 360 hours of radiation influence the model positively such that both the learning and non-learning accuracy increase.

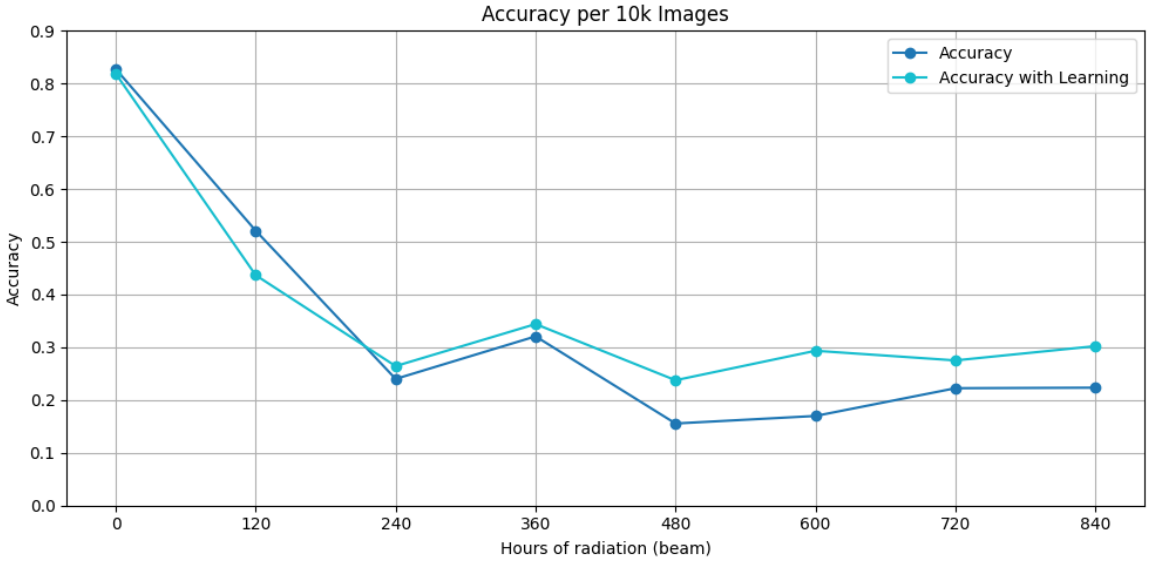


Figure 5.20: Accuracy over time for 10000 images with simulated radiation (2nd run)

In the third plot Figure 5.21 we still see the drop in accuracy after 120 hours, but then the learning keeps adjusting and it returns to an accuracy of 0.76 after it drops again, but now it drops less than the non-learning counterpart and the learning accuracy stays above the non-learning accuracy as time progresses.

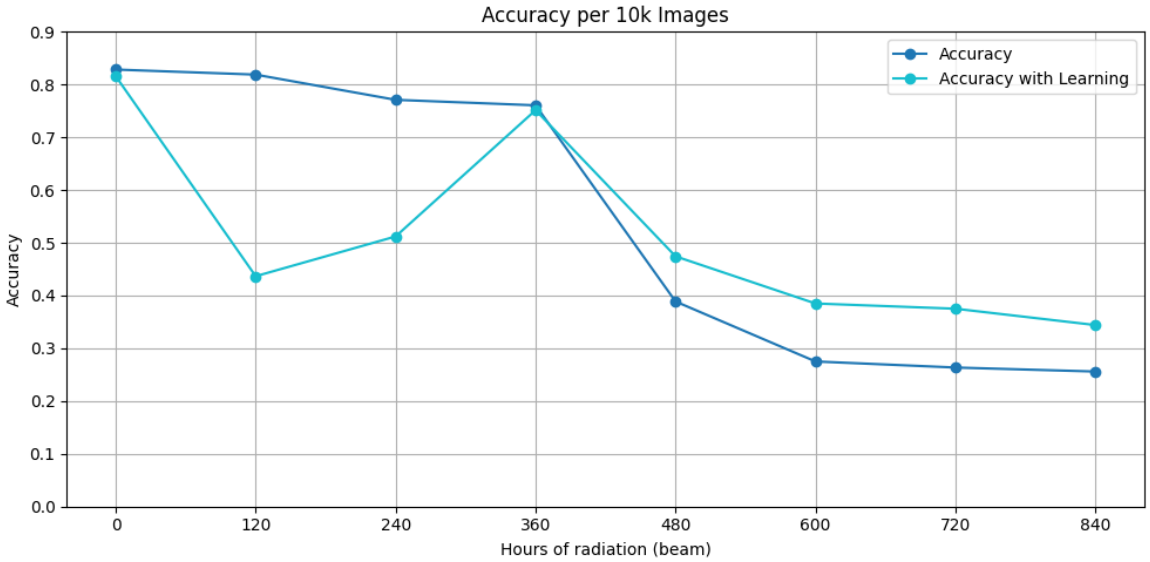


Figure 5.21: Accuracy over time for 10000 images with simulated radiation (3rd run)

Mean Time To Failure (MTTF) ODIN

The accuracy over time can also be used to calculate the Mean Time To Failure (MTTF) for the ODIN architecture. To determine the MTTF we have to define what we mean by a failing ODIN architecture. The working state of the ODIN architecture is measured by measuring the accuracy of the image interference task. The accuracy number tells us if the system is working. We can put the threshold of what we consider a failing ODIN architecture on different levels: we calculate the MTTF for falling below 75%, below 50%, below 25% and below 10%. With pure randomness, we would still expect something around 10%, because we have 10 classes.

An overview of the results for these accuracy threshold tests is given in Table 5.4. In the table, the equivalent of the radiation in years in space is given between the brackets. The accuracy column is about the average accuracy for all runs, except the last row, which tells when there exists at least one iteration with an accuracy lower than 10%. The two bottom rows of the table show the effect of learning, learning does not drop below 25% accuracy after 840 hours of simulation, when it does it is unknown because of the limited simulation data set. A more important finding is the last row, the accuracy never drops below 10% with learning, or at least not within 840 hours of simulated radiation, without learning the accuracy for individual runs already drops to 10% after 480 hours.

Accuracy	Without learning	With learning
$\forall\, x < 75\%$	120 (13610 years)	120 (13610 years)
$\forall\, x < 50\%$	240 (27220 years)	240 (27220 years)
$\forall\, x < 25\%$	840 (95270 years)	$>$*840 (95270 years)*
$\exists\, x < 10\%$	$>$480 (54440 years)	$>$*840 (95270 years)*

Table 5.4: Accuracy thresholds with and without learning

To get to those accuracy numbers we have to deal with the observation that there is not a hard limit on a number of bitflips which make the ODIN architecture fail, it really depends on where the bitflip happens. So we will utilize data gathered from the simulations with 10000 images, because there we have the exact same bitflips simulated for both the learning and non-learning configuration. We use averages from the radiation tests which have intervals of 120 hours to come up with the MTTF for ODIN. To get on average below the threshold of 75%, we need 120 hours of radiation. After that, both the learning and non-learning averages have dropped below 75% as shown in Figure 5.22.

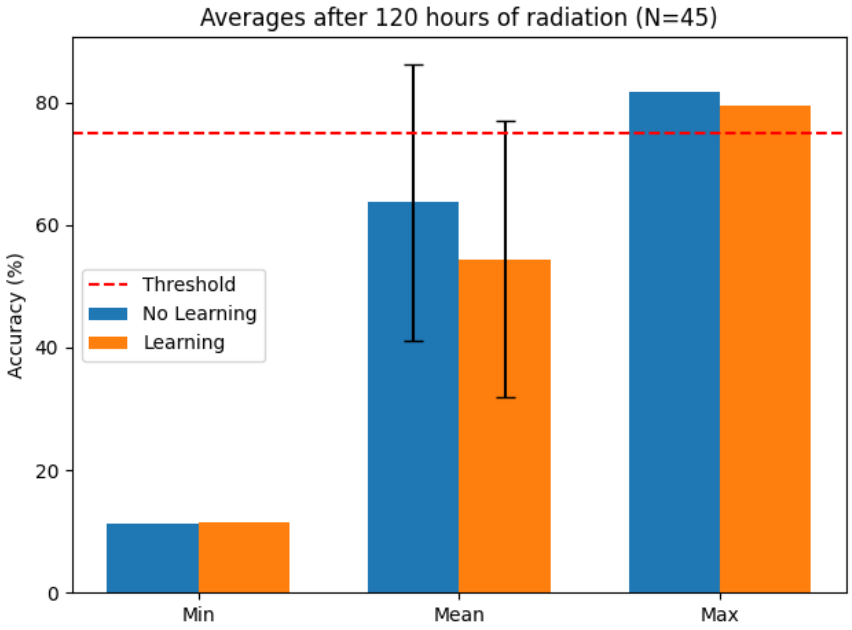


Figure 5.22: Accuracy after 120 hours of radiation

To get on average below the threshold of 50%, we need more hours of radiation. After 240 hours, the average accuracy drops below the threshold of 50% as shown in Figure 5.23.

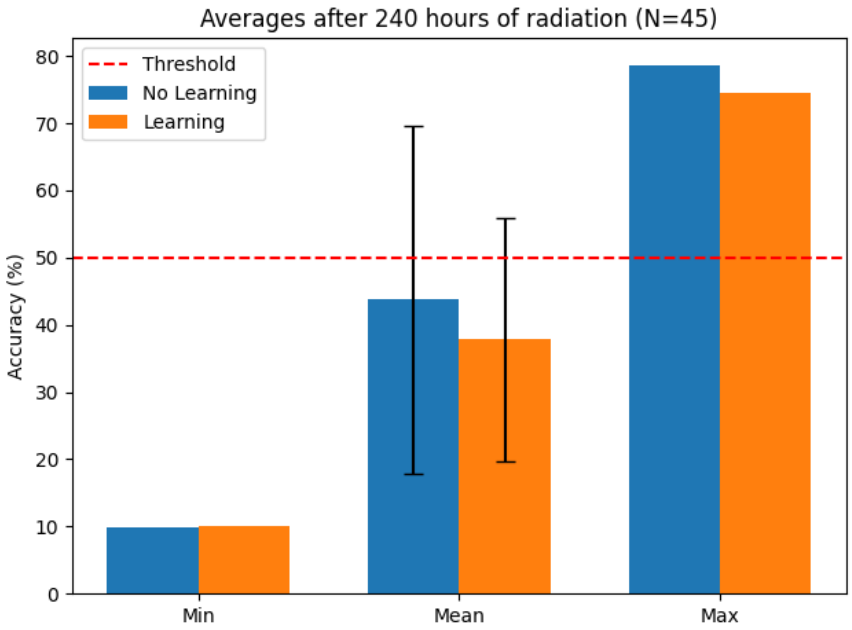


Figure 5.23: Accuracy after 240 hours of radiation

To get on average below the threshold of 25% we need 840 hours of radiation. But then only the non-learning configuration is below the threshold; see Figure 5.24. Learning keeps the accuracy stable around the 30% on average, so it will not drop below the threshold of 25% in our measurements.

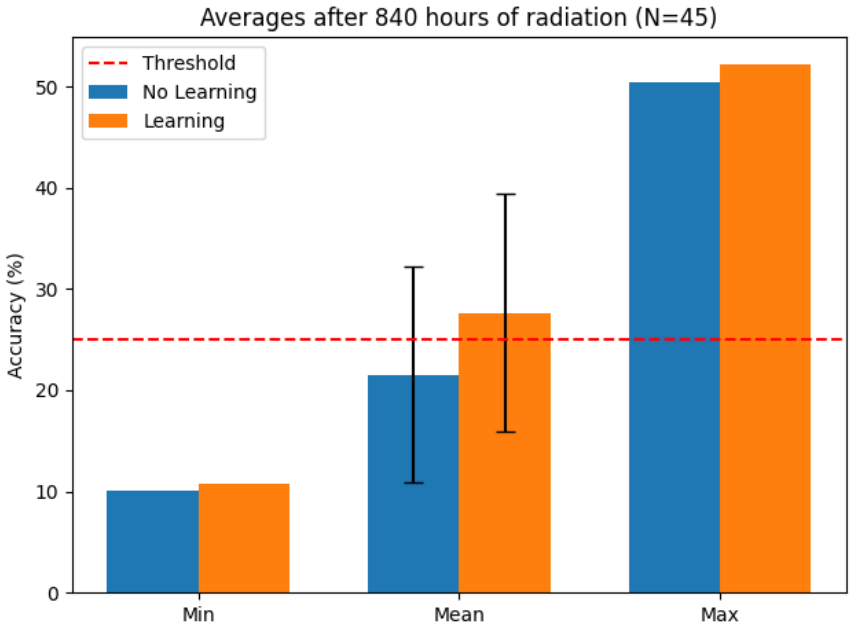


Figure 5.24: Accuracy after 840 hours of radiation

Within the samples we collected we did not get down to an average of 10% for either the learning or the non learning configurations, however for non learning configurations we see in Figure 5.26 that from 480 hours onward the minimum accuracy is around the 10%.

If we plot the average accuracies per period in one plot, we get Figure 5.25. We see that the learning accuracy initially drops down faster than the non-learning accuracy. But from the third period onward to the seventh period the learning accuracy is stabilizing but the non-learning accuracy keeps dropping down. This results in learning ending with a higher accuracy than non learning.

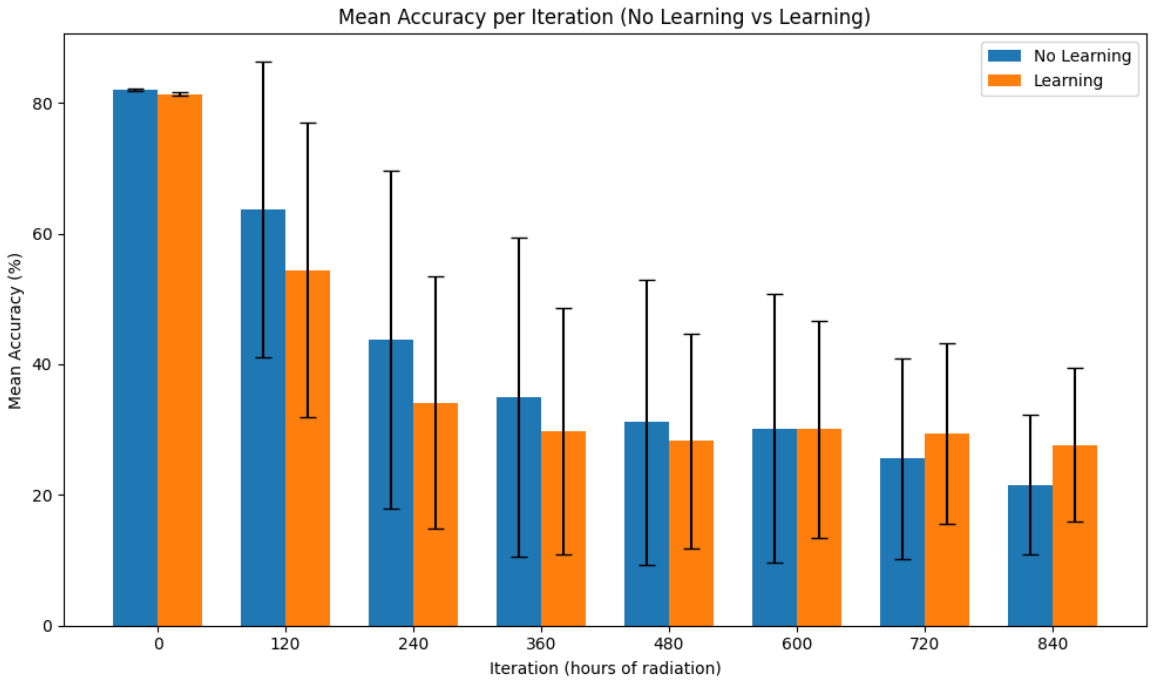


Figure 5.25: Accuracy per period of radiation

The initial drop in accuracy is also shown in the plots in subsection 5.2.4. The accuracy over time is also visualized in Figure 5.26, where it is evident that the accuracy under the learning scenario consistently maintains a higher minimum level compared to the non-learning scenario. However, during the interval between 360 and 480 hours of simulated radiation exposure, the accuracy of the learning configuration is observed to be notably low, although it exhibits partial recovery over time. At the last interval, it is observed that both the upper and lower bounds of the accuracy achieved with learning enabled surpass those obtained in the absence of learning.

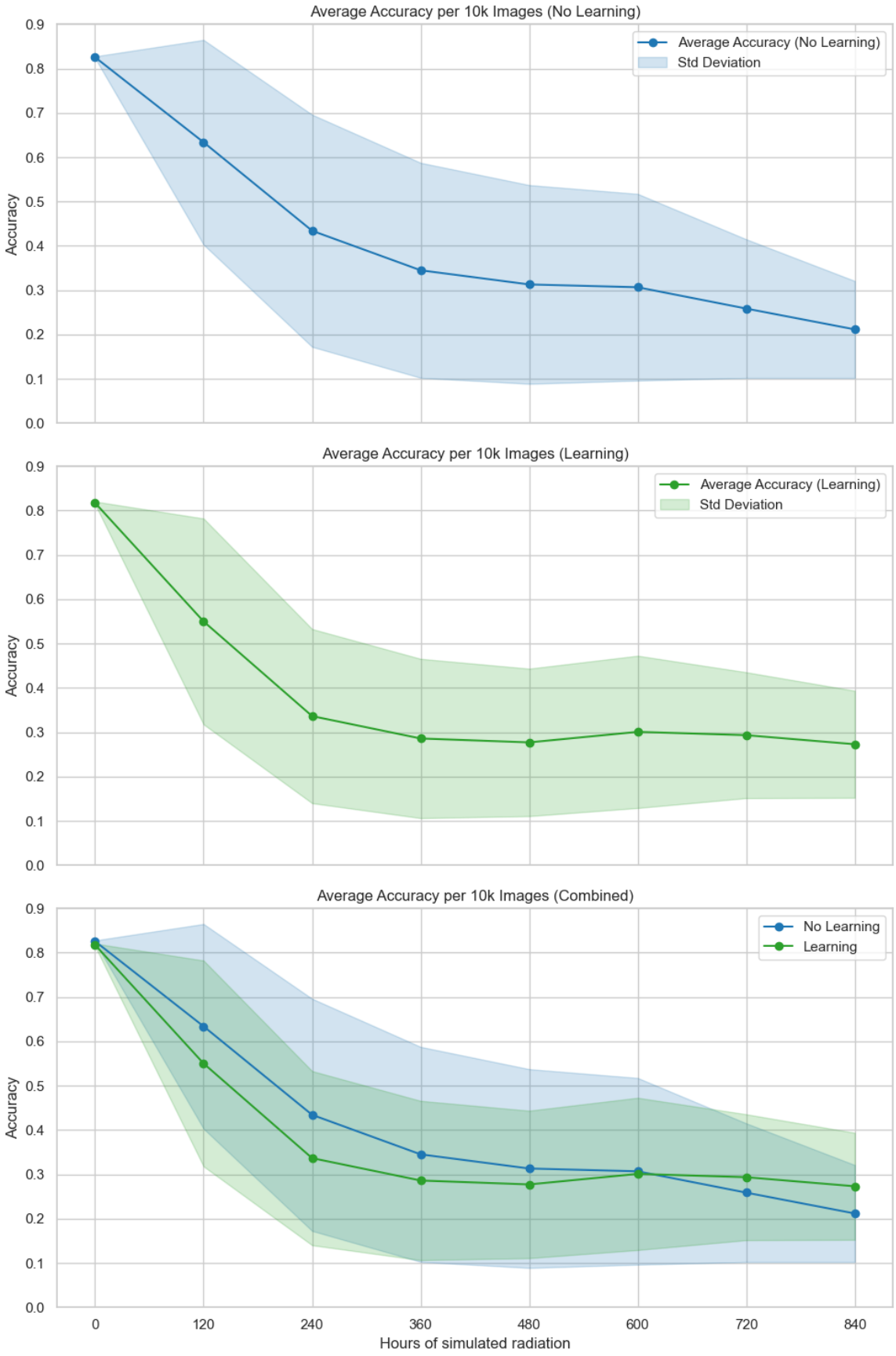


Figure 5.26: Average accuracy over time (n=45)

5.3 Faults

In this section, we look deeper into the faults caused by the radiation. The radiation beam that hits the FPGA in the radiation room causes those faults to occur. We measured the faults by measuring the number of bits changed for the synapse memory, which gives us insight into how radiation affected this memory area. Important to note here is that bits in the memory change for two reasons; on the one hand, the radiation causes them to change. Also, the learning mechanism, when enabled, has the ability to change the memory bits. That is why we use non-learning test runs for calculating the radiation effects, because for the runs without learning all the bitflips in the memory are caused by the radiation. From these radiation experiments, we find that the cross section is 4.32×10^{-9} cm^2. We also find that the MTTF for a single bitflip is 46 seconds. Because of the small chip usage of the ODIN architecture we have a MTTF for a relevant bitflip of 20 minutes. The total number of bitflips relates to the accuracy, a higher number of bitflips results in a lower accuracy.

Radiation beam time equivalents The radiation present in the beam is comparable to 10^9 times the atmospheric radiation[2]. The means that effectively for one device 1 second of beam time maps to almost 32 years of atmospheric radiation. Compared to space, we have approximately 10^6 times the space radiation at the beam [4]. This means that one second of beam time corresponds to 11.5 days of space radiation. See Table 5.5 for an overview of the mentioned proportions between the beam time and the environments in which the device can be placed.

Context	Beam Radiation Equivalent
Atmospheric radiation	10^9 times
Atmospheric lifetime	1 second = 32 years
Space radiation	10^6 times
Space lifetime	1 second = 11.5 days

Table 5.5: Radiation Equivalence Table

Measured bitflips When radiation affects the memory on the device and causes bit-flips, those bit-flips will stay in the memory and are easy to detect. Detect-

[2]See https://www.isis.stfc.ac.uk/Pages/Chipir.aspx

ing works as follows: start by reading the entire memory after the test run is finished, then compare this to the starting state and count the number bits which have changed. Doing this results in the findings shown in Table 5.6. The changed bits are shown in the column Avg(average) Bits. We also have a column only looking to the first 40 bits (the 10 x 4 bits of pre-configured weights), for this column we see that learning without radiation displays the same value for all the bits and those first 40 bits. With radiation on the numbers differ, because in the part outside the first 40 bits not that much learning is happening.

Images	Radiation	Learning	Avg Bits	Avg Bits (First 40)	Radiation (h:m:s)
6000	False	False	0.0	0.0	-
6000	False	True	280.68	280.68	-
60000	False	True	120.0	120.0	-
6000	True	False	28.94	0.53	00:22:53
6000	True	True	284.55	263.15	00:20:34
60000	True	True	350.8	129.2	03:48:34
10000	Simulation	True	2619.51	2619.51	840:00:00
10000	Simulation	False	2738.84	2738.84	840:00:00

Table 5.6: Measured bitflips with varying settings for radiation and learning.

Fault probability The fault probability is based on the average of the measured bit-flips in the 6000 images test cases. The memory bits can be divided into two sections for analysis purposes, one part contains the relevant bits (the bits holding the weights for the 10 enabled neurons) and the other part of the memory containing disabled neurons. To get a useful fault probability for simulation we only focus on the relevant part of the memory. For this we will calculated the MTTF for the bits in the relevant part. We define the MTTF as a memory bit being flipped, then it failed to retain its value. To get to the fault probability we will do the following; first, we calculate the fluence, then the cross section and from there we calculate the MTTF.

MTTF bitflips To get insight in how the radiation was influencing the device we analyzed the radiation data logs from the ChipIR facility. For this we used a tool called Radparser, see Appendix C. At ChipIR the flux is measured and logged. From this logging files we calculate the Fluence (Ψ) Equation 2.2, see Figure 5.27, that results in:

$$\Psi_{\text{total}} = 1.24 \times 10^{11} \text{ particles/cm}^2 \tag{5.1}$$

This fluence number is calculated for 7.0 hours. Based on the fluence we can calculate the flux (Φ) as follows, convert 7 hours to seconds $7 \times 60 \times 60 = 25200$ and use the inverse of Equation 2.2.

$$\Phi = \frac{\Phi_{\text{total}}}{t} = \frac{1.24 \times 10^{11}}{25200} \approx 4.92 \times 10^{6} \text{ particles/cm}^2\text{/s} \tag{5.2}$$

This number is very close to the ChipIr neutron flux measured by the facility [3]. The flux (Φ) is measured as:

$$\Phi = 5.0 \times 10^{6} \text{ particles/cm}^2\text{/s} \tag{5.3}$$

Our actual fluence number (Ψ) for the testing time should be a bit lower, because of some time was in there without running, 23348 seconds, which is ±6,5 hours. So we need to correct for that:

$$\Psi_{\text{corrected}} = \frac{\Psi_{\text{total}}}{\text{Time}_{\text{total}}} \cdot \text{Time}_{\text{executing}} \tag{5.4}$$

Then filling in the values we calculated already gives us:

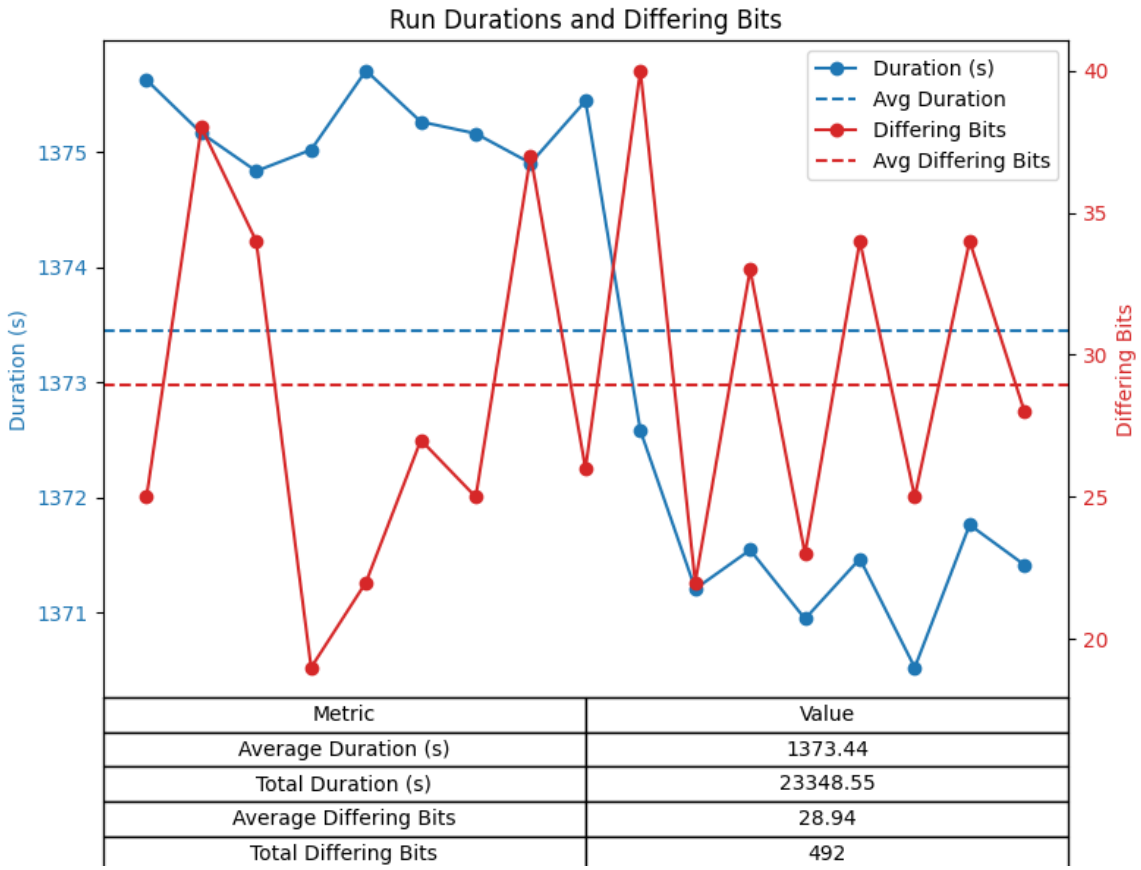
$$\Psi_{\text{corrected}} = \frac{1.24 \times 10^{11}}{25{,}200} \cdot 23{,}348 = \mathbf{1.14 \times 10^{11}} \tag{5.5}$$

```
------------------------ Polarfire ------------------------
    0.01 hours of beam off out of 7.0 hours
    Total Fluence = 1.24e+11
```

Figure 5.27: Fluence over the test period for the 6000 images without learning

For reference we use the tests for the 6000 images without learning, all the changed bits are bitflips in those tests, and they are thus caused by radiation. In Figure 5.28 we see that a total of 492 bits has changed.

[3] See https://www.isis.stfc.ac.uk/Pages/Chipir.aspx



Metric	Value
Average Duration (s)	1373.44
Total Duration (s)	23348.55
Average Differing Bits	28.94
Total Differing Bits	492

Figure 5.28: Radiation effects on 6000 images without learning

To calculate the cross section we use the calculated Fluence and the observed number of bitflips.

$$\sigma = \frac{\text{Number of observed bitflips}}{\text{Fluence}} \tag{5.6}$$

When we plug them into the formula of Equation 5.6 we get the following result:

$$\sigma = \frac{492}{1.14 \times 10^{11}} = 4.32 \times 10^{-9} \tag{5.7}$$

With those number we are able to calculate the MTTF, where we define a failure as a bitflip, with the formula from Equation 2.5.

$$\text{MTTF}_{\text{bitflip}} = \frac{1}{\lambda} = \frac{1}{\sigma \cdot \Phi} = \frac{1}{4.32 \times 10^{-9} \cdot 5.0 \times 10^{6}} = \frac{1}{0.0216} \approx 46\text{s} \tag{5.8}$$

Within the ODIN architecture we only used a small part. We only focus on relevant parts because there we are going to introduce simulated radiation. The relevant part of the synapses has the following size: Every synapse is connected to all 256 neurons, we are only interested in the first 10 neuron connections because those

are enabled, the others are disabled. So we get 256 synapses connected to 10 neurons each. Each weight is 4 bits, so we get $4 * 10 * 256 = 10240$ bits that are relevant. There are in total 256 neurons, which results in a total of $4 * 256 * 256 = 262144$ bits. The usage factor (η) is thus $\eta = 10240/262144 = 0.0390625 \approx 3.9\%$.

If we calculate the MTTF and define a failure as a bitflip in a relevant part we will get the following calculation:

$$\text{MTTF}_{\text{relevant bitflip}} = \frac{\text{MTTF}_{\text{bitflip}}}{\eta} = \frac{46}{0.039} \approx 1185\text{s} \approx 20 \text{ minutes} \tag{5.9}$$

This 20 minutes MTTF results in an approximate of 3 relevant bitflips per hour.

For the simulation with 10000 images we will use this bitflip rate. The results are shown in Table 5.6.

Bitflips and changed bits The radiation causes bitflips, however when measuring the number of changed bits we run into difficulties when we use learning. The learning is able to change bits as well. So for run with learning enabled only looking to changed bits does not tell the division of radiation induced bitflips compared to the total number of changed bits. To analyse this in more detail we depend on the measured changed bits after simulation runs, because for the simulation runs we exactly know how many bits we change with fault injection. Without learning the number of changed bits equals the number of injected faults (as long as we do not flip the same bit twice, but the chance of this happening is very small). For the learning we can just calculate the delta in changed bits between the learning and the non learning, they got the same faults injected, so all bits which are different between learning and non learning are caused by learning effects.

As shown in Table 5.6 we have an average of 2738 bits changed after radiation simulation without learning. With learning enabled we get an average over 2619 changed bits. If we calculate the average delta we get to 119 bits. This means that the learning at least reverts 119 bits back to their original state. Other memory changes resulting in 1 bit going back to the original state and one other bit changing could occur unnoticed, so there could be more bits changed by the learning.

In Figure 5.29 the accuracy is plotted against the number of changed bits, for clarity the number of changed bits starts at 2400 instead of 0, to make the differences better visible. From this graph, we see that the accuracy with learning is higher. Another interesting insight from those plots is that the number of bits changed is lower for the runs with learning. The spikes in the graph both downwards and upwards are not easily relatable to the number of changed bits, probably due to the 'hidden' bit changes, because the final number hides how many bits are really changed by learning. The delta in accuracy is compared to the delta in bits different in Figure 5.30.

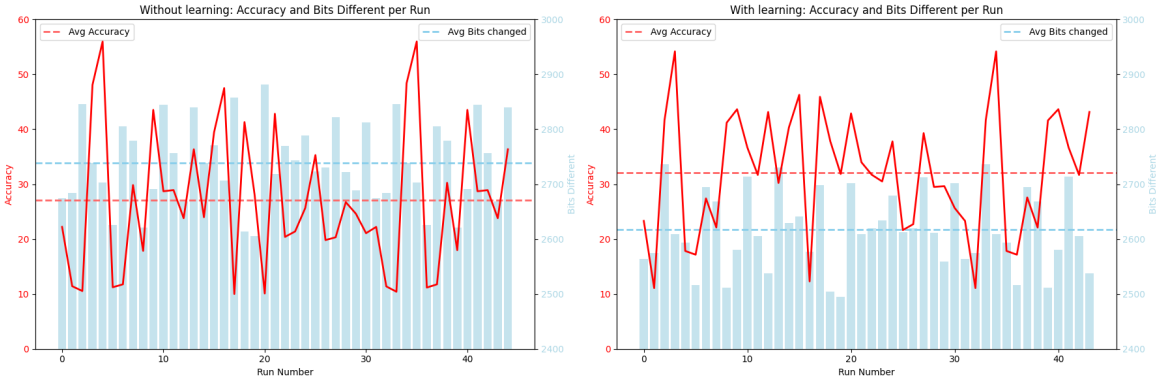


Figure 5.29: The number changed bits together with accuracy per test run
Left: without learning, right: with learning

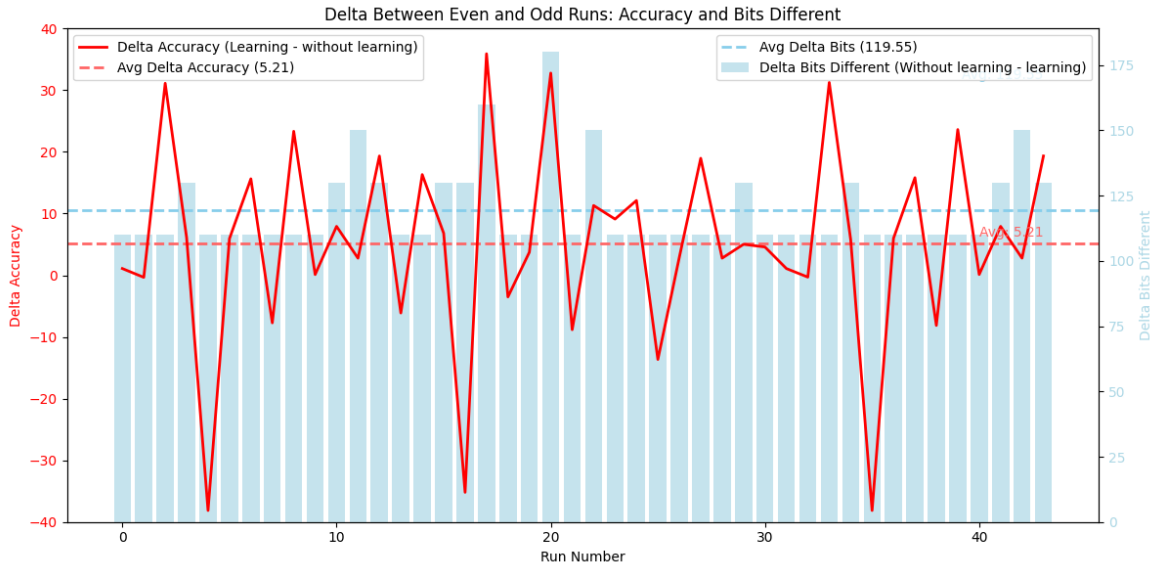


Figure 5.30: The delta in changed bits and accuracy per test run

Looking at Figure 5.30 the relation, on individual runs, between the delta in bitflips and the delta in accuracy does not show a consistent relation. For this data we cannot draw a conclusion about the delta of bitflips and the delta in accuracy, however the accuracy is higher with learning which has fewer changed bits, so overall there is some correlation, but on the individual runs we executed and analyzed the correlation is not clearly visible. So a lower number of changed bits, relative to the starting bits of the weights, gives us a higher accuracy, learning caused the number of changed bits to shrink, so the learning is helping by reverting bitflips and thus preventing the accuracy to drop when bitflips are injected.

5.4 Area usage

For the area usage of the ODIN architecture, we look at the results of the place and route functionality of the Libero tool, this gives insight into the [4], DFF [5], I/O [6] and logic element usage of the design. We will gather the data for 3 configurations. The first configuration is with learning and without full memory protection, this is what we tested under beam and also used for simulations. Then we measure the usage of the next configuration, which removes the learning part, this will give insight in how much hardware is required for the online learning. The last configuration is without learning, but with added protection for the memory, so TMR is used on both the synapse and neuron memory. Looking at the results in this section, we see that using online learning consumes much less area than using TMR on the entire memory. Learning requires 6381 DFF and 12797 4LUT, the TMR would use an additional 589824 DFFs for the memory (when no dedicated block RAM is used) + 158 DFFs for control, 92x more than learning requires. For the 4LUT usage the learning requires more 12797 instead of 2725, which is 4.6x as many.

Learning area usage The first configuration is the one we tested under the beam. Here, TMR is enabled for the parts that are not the SUT. The hardware required for online learning is also synthesized. With this configuration, we get the hardware usage numbers as shown in Table 5.7. This is the configuration that we used under the beam and also for simulation runs.

Table 5.7: Area usage with learning

Type	Used	Total	Percentage
4LUT	75,174	254,196	29.57 %
DFF	59,712	254,196	23.49 %
I/O Register	0	142	0.00 %
Logic Element	110,610	254,196	43.51 %
SRAM (KB)	36	17600	0.20 %

The second configuration is created to determine how much hardware is required for the online learning. We started with the same configuration as before, but now the learning logic is removed. The chip is synthesized again and that results in Table 5.8.

The hardware usage of the learning part is the delta between Table 5.7 and Table 5.8. This is given in Table 5.9. The learning does not add any I/O because it

[4] 4-bit lookup table
[5] Data Flip-Flop (1 bit)
[6] Input/output pins of the FPGA

Table 5.8: Area usage without learning

Type	Used	Total	Percentage
4LUT	62,377	254,196	24.54 %
DFF	53,331	254,196	20.98 %
I/O Register	0	142	0.00 %
Logic Element	96,848	254,196	38.10 %
SRAM (KB)	36	17600	0.20 %

is an internal system. It does not require more SRAM because it utilizes the already available SRAM. The number of 4LUT increases significantly, it adds +/-20% to the number of lookup tables. For the flip flops we see an increase of +/-12%. The number of logic elements also increases by +/-14%. So the learning part is quit substantial for the ODIN architecture.

Table 5.9: Extra area usage (online learning)

Type	Used	Total	Percentage
4LUT	12,797	254,196	5.03 %
DFF	6,381	254,196	2.51 %
I/O Register	0	142	0.00 %
Logic Element	13,762	254,196	5.41 %
SRAM (KB)	0	0	0.00 %

TMR area usage Another approach would be to remove learning and apply TMR to the memory of the ODIN architecture; the hardware requirements for this are shown in Table 5.10. The SRAM usage increases from 294,912 bits (36KB) to 884,736 bits (108KB) by doing this. The SRAM is added to the table because of the uSRAM and LSRAM block on the FPGA, those blocks are utilized for the SRAM, so the basic component resources like the 4LUT and DFF are not used for creating the memory.

Table 5.10: Area usage without online learning, with TMR on memory

Type	Used	Total	Percentage
4LUT	65,102	254,196	25.61 %
DFF	53,489	254,196	21.04 %
I/O Register	0	142	0.00 %
Logic Element	99,403	254,196	39.10 %
SRAM (KB)	108	17600	0.61 %

The delta in hardware resources needed with applying TMR to the entire system is given in Table 5.11. Applying TMR to both the neuron and synapse memory causes increases in resource usage, the largest change is in the SRAM usage, which is 200% higher. The 4LUT usage increases by +/-4%. The DFF usage sees a marginal increase of approximately 0.3%. Those changes require +/-3% more logic elements on the FPGA. If we would want to use the data flip flops instead of the SRAM, we would need one DFF per bit, for the additional 589,824 bits we have to store there are not enough DFFs available on the chip (only 254,196 DFFs are available).

Table 5.11: Extra area usage (TMR on synapse and neuron memory)

Type	**Used Difference**	**Total**	**Percentage**
4LUT	2725	254,196	1.07 %
DFF	158	254,196	0.00 %
I/O Register	0	142	0.00 %
Logic Element	2555	254,196	1.01 %
SRAM (KB)	72	17600	0.41 %

So the TMR on the entire ram will consume much more hardware than the additional hardware needed for the online learning.

Chapter 6

Discussion

In this chapter, we discuss the results and conclusions from chapter 5.

We first discuss the findings for all tests without learning. After that, we dive into the results with learning.

Without online learning Looking at the results for runs without online learning we see that the claims from the existing literature [7, 12, 21] that neuromorphic architectures are fault-resistant and thus radiation-resistant are correct.

In general, the system continues to behave very similar with respect to the radiation on the system. This is due to multiple reasons: First of all, the level of radiation does influence only a small percentage of all the memory bits. Within the memory, we did not utilize a large part because of the configured ODIN architecture with 10 neurons. This means that only 40 bits (4 bits per neuron) of each synapse memory space are utilized. The synapse memory space is 256*4=1024 bits large, so of all the memory, we only used a small part. This allows us to see radiation occurring on the memory without it affecting the interference accuracy significantly. Secondly, the small memory layout of the synapse makes it that the weight cannot change that much relatively speaking with one bitflip, because all the possible values are already used after the offline training, so no risk of a way larger number than the already available numbers. The memory layout for a synapse has 4 bits, the Most Significant Bit (MSB) is the enable bit, the other 3 bits are used for the weight. The impact of a bitflip is thus relative to the bit position, a bitflip on bit 0 (the Least significant Bit (LSB)) has the least impact, while a bitflip on bit 3 enables or disables the synapase for a specific input pixel. Finally, there are no compression layers, so all input pixels are connected to all neurons, a small change in the synapse for one pixel to neuron connection is then only present in that neuron and does not really influence the rest of the system. To make a neuron cause the classification to become incorrect means that that neuron not only should increment its number of output spikes, but it should increase it to be more than the correct neuron delivers.

However, at a certain point the ODIN architecture should fail completely. Because without any radiation protection, the changes in the memory will accumulate and should cause the classification accuracy to be as bad as random guessing. To reach this point, we needed simulation runs to prove when this point happens because the test data from the beam experiments are not enough. To get the ODIN architecture to fail completely, we needed a total of 480 hours of simulated radiation, equivalent to $5.4 \cdot 10^4$ years of radiation in space. This is a large number but it is imporant to keep in mind that we have a very small architecture, which classifies 16x16 pixel images. If we would do a classification on a Full HD input image (1920x1080) that would be 8100x larger than our current input size. Our system scales linearly with input size, so when the vulnerability scales linearly as well, which is likely, this would mean that we probably see complete failure within 7 years.

With online learning Looking at the results of the online learning, we notice the following: The learning makes the system divert a little bit of the pre-trained weights from python. This means that the weights are changed to a more stable state for online learning on the ODIN architecture. However, this learning reduces the accuracy a little bit. The effect of the learning quickly stabilizes and then keeps the accuracy around a constant deviation from the non-learning baseline. Then when running the 60000 images for an extended period of time we see 2 occurrences of a complete destruction of the classification accuracy; in those cases, the learning failed to correct for the radiation faults, and thus the performance degrades. Interestingly we saw the same accuracy drop happen in the simulation as well, where we have comparison data for the learning vs. non-learning with the exact same bitflips introduced. Here we see that the learning has some corrective power by keeping the performance drop way smaller than the drop occurring on the non-learning configuration. However, learning is not capable of maintaining accuracy around 80% average. On shorter runs, up to 360 hours, we see both learning and non-learning go down in accuracy, this indicates that the learning is not strong enough to remain at the high accuracy level. Looking at the plots for accuracy over time for the larger period intervals of 120 hours we see that the learning is overreacting on the first period of radiation, but then over time it recovers and end ups with higher accuracy than the non-learning. The learning is capable of preventing a full collapse of the accuracy, without learning we see instances were all the images are classified as one digit, but with learning this does not occur, this shows positive effects of learning. The drawback is that the most positive learning results are only visible in the low accuracy ranges, meaning that in the current form and with the currently configured parameters learning is not capable of maintaining high accuracy, but it is capable of preventing total system failure.

To get to the point of a completely failing ODIN architecture we do not know yet at which point in time that will happen, our learning simulation runs are not long enough to give us insight into this. This failure should mean that the accuracy drops to 10% which does not happen after 7 periods of 120 hours. This means that we have a time to failure larger than 840 hours of simulated radiation. When we calculate this number in years of radiation in space, we get $9.5 \cdot 10^4$ years. So we need more than $9.5 \cdot 10^4$ years of space radiation before the learning configuration could start to fail. If we consider the small input size of 16x16 images again and use the same example as used in the discussion section above, we have to increase the architecture size again by a factor of 8100 to be able to classify the full-HD input images. This gives us that the ODIN architecture with learning for this input size would not fail within +/-12 years. So, online learning gives us at least an increase of $>$175% for the time between failures.

Conclusion To conclude on radiation resistance of neuromorphic architectures it is safe to say that a small system like this ODIN architecture is already quit radiation resistant because we expect a time to failure in the order of $5.4 \cdot 10^4$ years.

To conclude on the effects of adding learning, we have the key takeaway that learning increases the time between failures with at least 175%.

Chapter 7

Conclusion

After collecting and analyzing all the research data we will now conclude the research by answering the research questions. The main research question is **To what degree and in what manner can neuromorphic architectures endure radiation exposure?** To answer that question, we have to answer the following sub-questions:

To what extent does the implementation of online learning during inference enhance the radiation resilience of neuromorphic architectures? The results show that incorporating online learning extends the time to failure by at least 175% compared to non-learning inference. In the short term, however, learning does not outperform non-learning due to overreacting to radiation induced changes, which limits its effectiveness under brief radiation exposure. Over extended exposure, learning sustains higher accuracy and prevents the ODIN architecture from collapsing to accuracy levels around 10%. These findings indicate that while online learning offers limited benefit in short durations, it significantly enhances radiation resilience over longer periods.

From this the first sub-question follows: *Can online learning mechanisms serve as effective substitutes for traditional error correction methods in neuromorphic architectures subjected to radiation exposure?* With our current implementation the learning is not strong enough to withstand the radiation effects in such a way that the accuracy is maintained. Traditional error correction methods like TMR should be able to maintain memory integrity and thus maintain the high accuracy. Overall, learning shows it ability of keeping accuracy from dropping down to 10%, however, this does not yet prove that learning can maintain the initial accuracy of +/-82%. For this further research is needed to make the learning more accurate and prevent the learning from overreacting.

The other sub-question is related to this is: *How do the spatial, power, and performance requirements for the implementation of online learning compare with those for traditional error correction mechanisms in neuromorphic chips?* Online learning

requires only a fraction of the hardware overhead of traditional redundancy-based protection. For example, implementing Triple Modular Redundancy (TMR) on the memory flip-flops demands up to 92× more DFFs than online learning. The additional memory and logic required by TMR also increase power consumption, while online learning avoids this overhead. In terms of performance, online learning introduces extra clock cycles to perform weight updates, whereas TMR mainly incurs a small latency from majority voting. Overall, online learning achieves resilience with far lower spatial and power costs than TMR, at the expense of some runtime overhead.

Which parts of a neuromorphic architecture are the most vulnerable to radiation influences? The architecture has multiple parts, which could all be influenced by radiation. However, we do not want to let the control get influenced by radiation, so all controlling logic is protected by TMR. This leaves the memory unprotected. We have two memory areas, one small memory area containing the neuron states and a large memory area containing the synapse weights. We saw that the most faults occur in the synapse weights memory, which occupies the largest area of the chip. The size is the reason why it is more likely to be influenced by radiation.

The first sub-question is: *How can we inject faults into the hardware?* For the fault injection we used two distinct methods. The first method was using a radiation beam, which was available for this research at the ChipIR at the Appleton Labratory. With this radiation beam we were able to get bitflips on the FPGA. A radiation test like this hits the entire chip, causing bitflips on all possible locations, that is why we had to introduce TMR on the parts which we needed for controlling the architecture, but which we did not want to test. The other method is injecting fault manually, for this we use the UART connection which is already available, which allows us to send commands to change the memory from the control system.

The next question asks the following: *How to monitor the number of faults occurring in the memory areas for the weights and neurons states?* For this we use the UART communication as well, at the end of test run we readout the entire memory of the FPGA, we store this and compare it with the initial memory configuration we sent to the FPGA before starting the test. The differences between those two memory dumps tell us the number of changed bits.

The last sub-question is: *How vulnerable are spikes data-structures to faults?* We did not see errors in the spike data-structures; however, it will require additional steps to measure this accurately. In our testing, it does not seem to have spike faults; the faults were mostly in the synapse memory, because this is the largest vulnerable part of the implementation.

How can radiation tolerance be improved with training the model in a different way? We applied quantization to get the model to fit. This was required because of the small memory size for the weights per synapse. A small memory footprint also helps against radiation, the cross section we calculated gave the radiation vulnerability per cm^2, which tells us that with a smaller area the device is less vulnerable. So, any training method that reduces the chip area increases radiation tolerance.

This brings us to the sub-question: *Is quantization a feasible technique to improve radiation tolerance?* Quantization reduces the memory footprint. A smaller chip will get fewer bitflips and thus is more radiation resilient. So quantization is helping with radiation resistance.

The last sub-question to answer is: *Is a sparser network more resistant to radiation?* Based on the measurement of the bitflips in the memory areas we can conclude that using a small percentage of the memory area makes it more likely to get fewer bitflips. However, the vulnerability could still be equal, because when the absolute number of relevant bitflips has decreased, the impact per bitflip will be higher because of the sparser network.

We finish with the conclusion for the main question: **To what degree and in what manner can neuromorphic architectures endure radiation exposure?** Neuromorphic architectures can endure radiation exposure to a meaningful degree, but their resilience depends strongly on the size of the memory and the duration of radiation exposure. The most vulnerable component is the synapse weight memory.

Online learning during inference extends the operational lifetime under radiation by more than 175% and prevents the complete collapse of accuracy that we saw without learning. However, learning overreacts to faults and cannot yet maintain the initial accuracy levels of approximately 82%. Traditional error correction mechanisms such as TMR provide stronger guarantees of memory integrity, though at the cost of much higher spatial and power requirements. On the contrary, online learning achieves fault mitigation with a far lower hardware overhead, incurring only moderate runtime cost.

Radiation resilience can be further enhanced by architectural and training choices that reduce the memory footprint, such as quantization, which lowers the bit-flip rate by shrinking the vulnerable area.

In general, neuromorphic architectures can tolerate radiation exposure through a combination of redundancy, online learning, and reduced memory footprint. Although online learning alone does not yet have the capacity to completely replace traditional protection mechanisms, it provides a lightweight resilience strategy that, when combined with further improvements in learning, quantization, and selective redundancy, enables sustained functionality under extended radiation exposure.

Chapter 8

Future work

Based on the findings and the progress we made, we stumbled upon a lot of things that need further research; because of the limited time, many of them are left out of scope for the current work, but will be mentioned in this section.

Online learning The main bottleneck for this research was the learning performance; Further research to prevent overreacting in the learning mechanism and a better understanding of the effects of the learning parameters have the potential to improve this work further. We need to prevent overreacting to keep the accuracy high for the shorter test runs, the results we saw with online learning look promising, but to fully benefit from the online learning we need it to be more reliable and not causing the accuracy to drop initially faster than without learning. The learning depends on a configuration, where multiple parameters can be configured, an extensive search to all possible combinations could be insightful and could result in new insights. To do this effectively, additional automation of this process is preferable, which directly opens the door to additional future work. A faster and more automated execution of the different test cases will also help speed up further research.

Model training The model training was initially performed with Python, after quantization the accuracy has dropped to 82%. Additional research can be carried out to try to get the initial accuracy higher, because this can also give benefits for monitoring learning behavior. Higher accuracy makes the initial weights more important, and deviations of those initial weights will most likely cause the accuracy to drop. With a lower starting accuracy the chance that a bitflip increases the accuracy becomes larger. So further research to do quantization-aware model training and convert the model to the ODIN architecture without losing accuracy would be a meaningful improvement.

Testing on other architectures With the ODIN architecture we have shown positive learning effects; to continue the research on learning effects on neuromorphic architectures, it is good to try it on different architectures as well; ReckON is, for example, a good candidate, because of the larger size, which allows for more complex tasks. An alternative strategy involves developing a fundamental architecture independently and integrating learning mechanisms therein, thereby allowing for complete adaptability in the implementation of learning protocols to ensure radiation resistance.

Memory footprint reduction An investigation into memory reduction by further quantization or sparsification of the architecture can be interesting as well, reducing the memory footprint could reduce the radiation vulnerability as well. However, because of the increased relevance of the remaining parts, it is uncertain whether this really increases the radiation resistance, making this a very interesting research topic. This will require a thorough validation and testing before any meaningful claims can be made.

Testing In order to make testing faster and more effective, the automation level of the testing should be increased as much as possible. This means that executing many tests will become only a matter of time. Testing time can be reduced through parallelization, e.g., instantiating multiple ODIN cores per FPGA or distributing across multiple FPGAs. This makes it easier to do more tests, which will make the results more statistically significant.

Monitoring In terms of monitoring, there is room for improvement. Currently, we compare the memory at the end of the test with the initial memory state, which gives us insight into how many bits have changed. But this does not tell us when bits have changed. It also does not distinguish between bits changed by learning or by radiation. Expanding the FPGA functionality by recording the memory changes done by learning can make the distinguishing between radiation bitflips and memory changes by learning much easier, which will result in more relevant data for the memory analysis. Another improvement is to collect the memory data more often, not only at the end of the run, but also during the run, which gives more insight into the distribution of bitflips over time.

Model size Another interesting topic is the model size; does the radiation vulnerability scale linearly with the model size? Future work should investigate how radiation vulnerability scales with size; for example, we could scale up from 16x16 to 28x28

bit images. Multiple different model sizes could be tested for this to get better insight into the consequences of scaling a neuromorphic chip.

Data collection With more data, it is easier to see trends and it also makes the analysis on the data more statistically significant. So, scaling up data collection is also a good further research direction. This can start as simple as doing an additional radiation test with the current system to collect more information on the behavior of the system when exposed to radiation. The collection of radiation beam data is also crucial to validate simulation assumptions.

Conclusion Looking at all the points mentioned above, there is much work to be done. A good starting point is the future work on improvements to online learning, this will bring us closer to an answer to the question does online learning enhance radiation resilience of neuromorphic architectures?

Bibliography

[1] Filipp Akopyan et al. “TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (2015), pp. 1537–1557. DOI: 10.1109/TCAD.2015.2474396.

[2] Rick Alena. *Error Resilient Neuromorphic Systems Using Embedded Predictive Neuron Checks*. Technical Memorandum NASA/TM–20220011806. NASA, 2022. URL: https://ntrs.nasa.gov/citations/20220011806.

[3] Chandramouli Amarnath and Abhijit Chatterjee. “Error Resilient Neuromorphic Systems Using Embedded Predictive Neuron Checks”. In: *2023 IEEE 24th Latin American Test Symposium (LATS)*. 2023, pp. 1–2. DOI: 10.1109/LATS58125.2023.10154498.

[4] C. Andreani et al. “Fast neutron irradiation tests of flash memories used in space environment at the ISIS spallation neutron source”. In: *AIP Advances* 8.2 (2018), p. 025013. DOI: 10.1063/1.5017945.

[5] Hajar Asgari, Babak Mazloom-Nezhad Maybodi, and Yulia Sandamirskaya. “Digital multiplier-less implementation of high-precision SDSP and synaptic strength-based STDP”. In: *International Journal of Circuit Theory and Applications* 48.5 (2020), pp. 724–738. DOI: https://doi.org/10.1002/cta.2753. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cta.2753. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cta.2753.

[6] Rajan Bedi. “A comparison of space-grade FPGAs, Part 1”. In: *EDN* (June 2014). Accessed: 2025-07-17.

[7] Rachel M. Brewer et al. “The Impact of Proton-Induced Single Events on Image Classification in a Neuromorphic Computing Architecture”. In: *IEEE Transactions on Nuclear Science* 67.1 (2020), pp. 108–115. DOI: 10.1109/TNS.2019.2957477.

[8] Alfio Di Mauro et al. “SNE: an Energy-Proportional Digital Accelerator for Sparse Event-Based Convolutions”. In: *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2022, pp. 825–830. DOI: 10.23919/DATE54114.2022.9774552.

[9] E. Farquhar, C. Gordon, and P. Hasler. “A field programmable neural array”. In: *2006 IEEE International Symposium on Circuits and Systems*. 2006, 4 pp.–4117. DOI: 10.1109/ISCAS.2006.1693534.

[10] Bruno Forlin et al. “From Ground to Orbit: A Robust and Efficient Test Methodology for RISC-V Soft-Cores”. In: *IEEE Transactions on Device and Materials Reliability* 25.1 (2025), pp. 27–36. DOI: 10.1109/TDMR.2025.3537718.

[11] Charlotte Frenkel and Giacomo Indiveri. “ReckOn: A 28nm Sub-mm2 Task-Agnostic Spiking Recurrent Neural Network Processor Enabling On-Chip Learning over Second-Long Timescales”. In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65. 2022, pp. 1–3. DOI: 10.1109/ISSCC42614.2022.9731734.

[12] Charlotte Frenkel et al. “A 0.086-mm^2 12.7-pJ/SOP 64k-Synapse 256-Neuron Online-Learning Digital Spiking Neuromorphic Processor in 28-nm CMOS”. In: *IEEE Transactions on Biomedical Circuits and Systems* 13.1 (2019), pp. 145–158. DOI: 10.1109/TBCAS.2018.2880425.

[13] Charlotte Frenkel et al. “A fully-synthesized 20-gate digital spike-based synapse with embedded online learning”. In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2017, pp. 1–4. DOI: 10.1109/ISCAS.2017.8050219.

[14] Sahil Hassan, Parker Dattilo, and Ali Akoglu. “A Novel Implementation Methodology for Error Correction Codes on a Neuromorphic Architecture”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2023). ISSN: 0278-0070. DOI: 10.1109/tcad.2023.3285410. URL: http://hdl.handle.net/10150/672270.

[15] Jin-Hyuk Kong et al. “Cognitive model of emotion and the noetic-endetic distinction”. In: *Frontiers in Psychology* 3 (2012), p. 242. DOI: 10.3389/fpsyg.2012.00242. URL: https://pubmed.ncbi.nlm.nih.gov/22807913/.

[16] Joshua Mack et al. “RANC: Reconfigurable Architecture for Neuromorphic Computing”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.11 (2021), pp. 2265–2278. DOI: 10.1109/TCAD.2020.3038151.

[17] Ralph C Merkle. “Energy limits to the computational power of the human brain”. In: *Foresight Update 6* (2007).

[18] Farhad Modaresi, Matthew Guthaus, and Jason K. Eshraghian. “OpenSpike: An OpenRAM SNN Accelerator”. In: *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2023, pp. 1–5. DOI: 10.1109/ISCAS46773.2023.10182182.

[19] Maarten Molendijk et al. “Benchmarking the Epiphany Processor as a Reference Neuromorphic Architecture”. In: Aug. 2023, pp. 21–34. ISBN: 9781003377382. DOI: 10.1201/9781003377382-2.

[20] L.H. Mutuel. *Single Event Effects Mitigation Techniques Report*. Technical Report DOT/FAA/TC-15/62. Performing Organization: Thales Avionics, Inc.; Contract No. DTFACT-13-D-0008. Atlantic City International Airport, NJ: Federal Aviation Administration, William J. Hughes Technical Center, Feb. 2016, p. 296.

[21] Jonathan Naoukin, Murat Isik, and Karn Tiwari. “A Survey Examining Neuromorphic Architecture in Space and Challenges from Radiation”. In: *arXiv preprint arXiv:2311.15006* (2023). Submitted to IEEE Journal on Miniaturization for Air and Space Systems. URL: https://arxiv.org/pdf/2311.15006.

[22] Open Neuromorphic Community. *open-neuromorphic: A list of neuromorphic software projects*. https://github.com/open-neuromorphic/open-neuromorphic. GitHub repository, accessed 17 July 2025. 2025.

[23] Heather Quinn. “Radiation effects in reconfigurable FPGAs”. In: *Semiconductor Science and Technology* 32.4 (Mar. 2017), p. 044001. DOI: 10.1088/1361-6641/aa57f6. URL: https://dx.doi.org/10.1088/1361-6641/aa57f6.

[24] Catherine D. Schuman et al. “Publisher Correction: Opportunities for neuromorphic computing algorithms and applications”. In: *Nature Computational Science* 2.3 (2022), pp. 205–205. ISSN: 2662-8457. DOI: 10.1038/s43588-022-00223-2. URL: https://doi.org/10.1038/s43588-022-00223-2.

[25] Ronald Scrofano, Scott C. Davis, and Jennifer L. Taggart. “Radiation Test Performance of the Intel Loihi Neuromorphic Processor”. In: *2024 IEEE Space Computing Conference (SCC)*. 2024, pp. 86–92. DOI: 10.1109/SCC61854.2024.00016.

[26] Yuanfu Zhao Suge Yue Xinyuan Zhao Shijin Lu Qiang Bian Liang Wang Yongshu Sun. “Single event soft error in advanced integrated circuit”. In: *Journal of Semiconductors* 36.11 (2015), p. 111001. ISSN: 1674-4926. DOI: 10.1088/1674-4926/36/11/111001. URL: https://www.jos.ac.cn/en/article/doi/10.1088/1674-4926/36/11/111001.

[27] Shiqiang Zhu et al. “Intelligent Computing: The Latest Advances, Challenges, and Future”. In: *Intelligent Computing* 2 (2023), p. 0006. DOI: 10.34133/icomputing.0006. eprint: https://spj.science.org/doi/pdf/10.34133/icomputing.0006. URL: https://spj.science.org/doi/abs/10.34133/icomputing.0006.

Appendix A

Used tools and created artifacts

I used the following software platforms/tools for my thesis project:

- Microsoft Windows 11 & Ubuntu 24.04.2
- Microsoft Visual Studio 2022 (with GitHub Copilot)
- Pycharm Professional 2024.3.5 (with GitHub Copilot)
- Visual Studio Code (with GitHub Copilot)
- Libero SoC v2024.2
- Git

AI usage statement During the preparation of this work, I used GitHub copilot for AI assisted programming. I used ChatGPT to search the Internet and help with programming. After using this tool/service, we thoroughly reviewed and edited the content as needed, taking full responsibility for the final outcome.

Artifacts All the artifacts relevant for this thesis, such as the Verilog code, the scripts, and all the collected logging files can be found at https://gitlab.utwente.nl/dcs-group/socs/polarfire-odin.

Appendix B

Radhelper

Radhelper is a tool created by the DCS group to control and monitor devices when they are tested under radiation. This tool can be found at https://gitlab.utwente.nl/dcs-group/radiation-setups/radhelper-embedded. For this research, we expanded the tool to add functionality to send data to devices. For data sending, we use a csv file with commands and data.

The file used for data sending contains lines with a command number, the command payload and a label like this:

```
50,00,255
60,01000000000000000000000000,255
66,0000CCCCCCCCCC000000000000000000000000000000000...0000,255
70,00000000000000000010002060204000000000000606040...0000,5
72,00000000601000000000001070000000000000003020000...0000,5
60,01000000000000000000000000,255
64,00,255
68,8001,255
```

The "..." represent left out digits because of too few space for long lines on this page

The Radhelper tool creates log lines as follows:

```
2025-05-03 18:02:40,969 [    INFO] [ODIN_BEAM_ROOM] Parsed
   13048 lines from ./../ODIN-testdata/
   commands_6000_829_no_learning.csv
2025-05-03 18:02:40,970 [   DEBUG] [ODIN_BEAM_ROOM] Handling
   input data[0]: 50, 1, 255
2025-05-03 18:02:41,023 [   DEBUG] [ODIN_BEAM_ROOM] Write:
   AA320100E1F055 (7)
2025-05-03 18:03:09,416 [    INFO] [ODIN_BEAM_ROOM] IMAGE CLASS
   : 1->1 [True]: [77, 80, 79, 79, 78, 78, 78, 78, 78, 79] (981)
2025-05-03 18:03:09,416 [    INFO] [ODIN_BEAM_ROOM] Accuracy
   [100]: 0.9 True: 90, False: 10
```

Appendix C

Radparser

The Radparser tool calculates the Fluence based on the logging files from the ChipIR facility. It was provided by the DCS group so that we could analyze the logging files from the radiation experiments. The Radparser tool can be found at https://gitlab.utwente.nl/dcs-group/radiation-setups/radparser.

We provide the tool with the following distance file (in csv format):

board	distance	start	end	facility_factor
Carier /w 6	40	0	0	1.82E+05
Carier /w 6	40	0	0	1.82E+05
Carier /w 6	40	0	0	1.82E+05
Carier /w 6	40	0	0	1.82E+05
Carier /w 6	40	0	0	1.82E+05
Carier /w 6	40	0	0	1.82E+05
Carier /w 4	67	0	0	1.82E+05
Carier /w 4	67	0	0	1.82E+05
Carier /w 4	67	0	0	1.82E+05
Carier /w 4	67	0	0	1.82E+05
Polarfire	100	03/05/2025 10:00:00	05/05/2025 12:30:00	1.82E+05

The output of the Radparser looks like this:

```
------------------------- Polarfire -------------------------
        1.02 hours of beam off out of 44.3 hours
        Total Fluence = 7.75e+11
```

Appendix D

MNIST

The MNIST dataset contains 70000 images, we used 60000 training images and 10000 test images. We also have smaller test sets to safe time when doing intef-erence and radiation testing. In section 4.3 the image size conversion from 28x28 to 16x16 is already explained. In this appendix we focus on describing the dataset distribution which causes the accuracy in section 5.2 to drop around 1500 images. This is due to unbalances in the dataset; in Figure D.1 we see that not every class is equally often represented. Around 1500 images we see that the digits 4, 7 and 9 are overrepresented, but those digits are also classified worser than average, they occur here much more than average, so the accuracy drops down around the 1500 images.

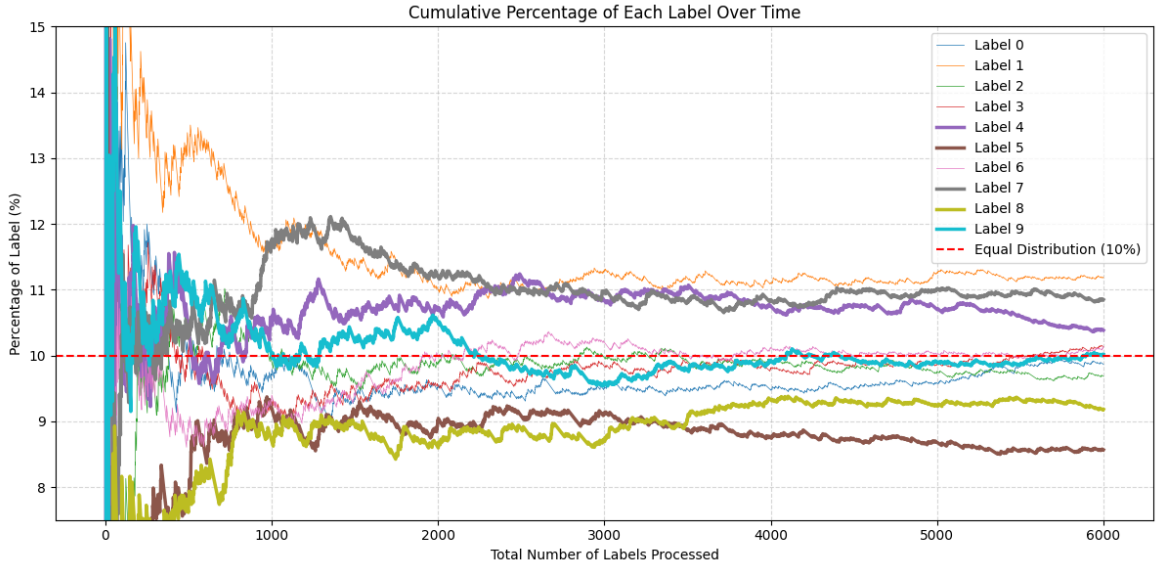


Figure D.1: The distribution per class over time in MNIST dataset for the first 6000 images

Eventually after all the 6000 images have been presented, the distribution per class is shown in Figure D.2, this shows that not all digits are equally often in the

dataset, which impacts our accuracy at certain points in the dataset.

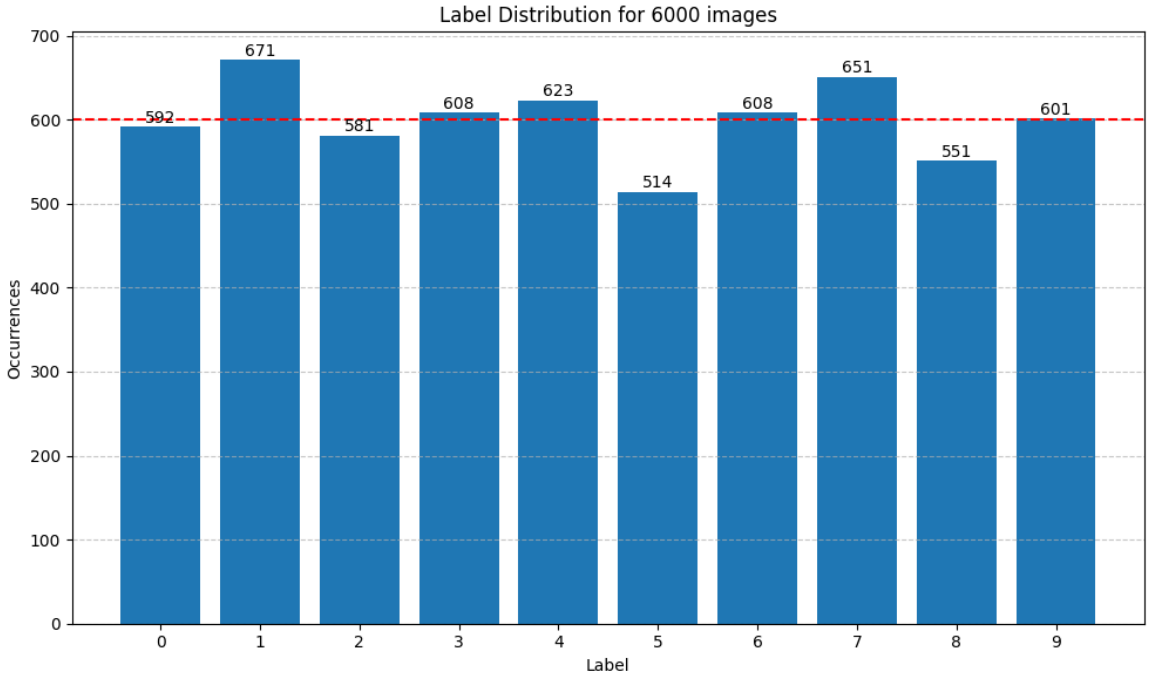


Figure D.2: The distribution per class in MNIST dataset for the first 6000 images

When looking at a larger portion of the dataset, for instance 60000 images, the distribution looks as follows in Figure D.3. It has become more balanced, but still not every digit occurs equally often.

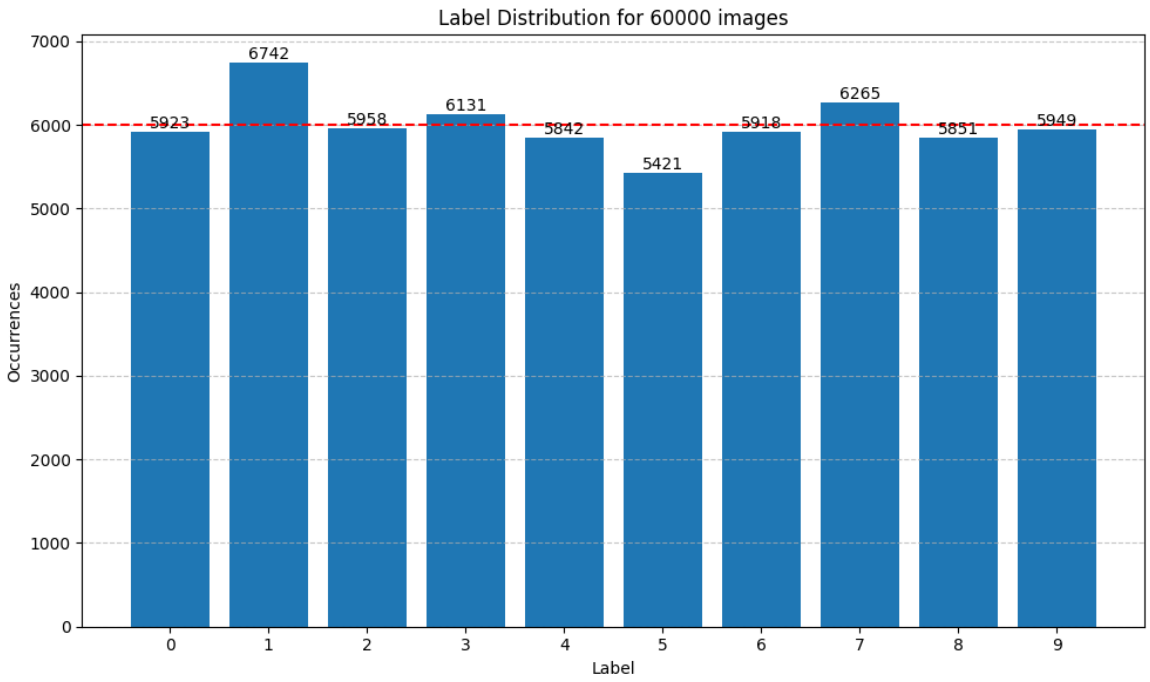


Figure D.3: The distribution per class in MNIST dataset for 60000 images