CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Anti-Aging Accelerators

A study on the effects of GPU degradation in AI applications

Master's thesis in Computer science and engineering

Björn Forssén

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Anti-Aging Accelerators

A study on the effects of GPU degradation in AI applications

Björn Forssén

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Björn Forssén

Supervisor & Examiner: Pedro Petersen Moura Trancoso, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Anti-Aging Accelerators
A study on the effects of GPU degradation in AI applications
Björn Forssén
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

By using a model based on first principles, it is possible to argue about the sustainability effects of aging AI accelerator hardware by relaxing some constraints. But first order principles have its limits and despite this effort to reason regarding the replacement of AI accelerator hardware and significant remains to be explored. This work presents some common considerations for reasoning about sustainability with computer hardware, some of the considerations being based on current trends are based on the situation at the time of writing. Then a model based on first principles is explained and is used to reason about replacement of AI accelerator hardware, this is combined with software simulation of faulty hardware in-place of faulty hardware. Finally, we present some of the considerations learned from testing the first-order model and conclude on some suggestions for further work to study the sustainability of AI accelerators.

Keywords: Computer, science, computer science, engineering, project, thesis, DNN, reliability, carbon footprint, sustainability.

Acknowledgements

I wish to thank my supervisor and examiner Pedro Petersen Moura Trancoso for his patient guidance and support. Additionally, many thanks to my family for their continued encouragement and support.

Björn Forssén, Gothenburg, 2026-02-01

Contents

List of Figures

1

Introduction

Since the start of the AI boom the interest in Deep Neural Networks (DNNs) has kept increasing, and while early DNNs relied on Graphical Processing Units (GPUs) for computing power the more modern DNNs can leverage specialized accelerators for even better performance. This in conjunction with the fact that with more AI accelerators being announced and released each year, it is safe to say that the demand and usage of accelerator hardware remains high and is likely to continue for some time [1].

With this increasing interest, there are both more DNNs being trained and more hardware needed to support it [2]. The increase in users of DNNs means more hardware units, more power to support the user base etc. Also, the hardware itself is having an increasingly larger effect on the environment with more complex production processes requiring rare materials to be used in power intensive processing steps. These factors combine to set an upwards trend for the carbon footprint of the Information and Communication Technology (ICT) sector which is estimated on the high end to be 2,1% to 3,9% of the Global Greenhouse Gas (GHG) emissions [3]. A ratio which is not massive in and of itself but by being connected to other more polluting sectors it is important that the carbon footprint trend is kept in mind.

With the objective of keeping the carbon footprint in mind for future developments of AI accelerators we propose an extension to the model presented by Eeckhout [4]. The extension takes advantage of DNN's ability to resist errors in order to mitigate some amount of degradation in the accelerator hardware allowing users to reduce the number of times they need to replace their hardware thus lowering the demand for accelerator hardware. This error resilience is similar to how synapses in the human brain is able to tolerate noisy signals, the effect is limited however and is determined by network design which may introduce spare capacity in the model to reach the correct answer even given some errors [5]. In order to implement this extension, we propose some new variables and discuss some scenarios to demonstrate the potential effect of an altered replacement policy. For some of the variables proposed we tried to approximate using available data but we also attempted to quantify the drop in application accuracy from hardware errors by using pre-trained version of ResNet18 and AlexNet for image classification on the ImageNet Large Scale Visual Recognition Challenge 2012 dataset [6].

1.1 Research questions

1. Accelerator hardware is designed to last for a specific amount of time, how does it start to behave towards the end of its lifetime and beyond?
2. When the hardware starts causing errors what effects does this have on AI applications?
3. For larger users of accelerator hardware, the decision to replace ageing hardware is not a simple decision. With respect to environmental impact, what are some of the most important considerations?

1.2 Limitations

Due to the availability of hardware and simulator software this paper focuses on the usage of GPUs in AI applications. This excludes hardware such as FPGA accelerators, NPUs and processors designed for AI applications, as a result different patterns or limits of degradation dependent on hardware type is not considered. The only application considered is image recognition due to the availability of test data and the limit of computational power, this means that the resilience of different networks based on application type are not considered.

1.3 Ethics considerations

The energy usage of DNN's can be significant, but since this project uses common pretrained models the resource cost of training is divided among many users and together with a limited amount of inference the energy usage of this project is kept within excessive limits.

Although, the ILSVRC2012 dataset is quite old and does not necessarily live up to recent standards since it is only used to determine model degradation through inference there should be minimal disruption from the dataset, despite its age.

2

Background

In this chapter we present some common considerations when reasoning about the sustainability of computer hardware. This includes the footprint occurred from production and usage represented here and later with embodied and operational footprint. We also present and discuss some of the ways to calculate emissions alongside some considerations for uncertainty in the data, but also some overarching trends.

We also present the carbon footprint model and the proposed extension to explore different options for replacement policies. As a part of this proposed extension there are several new variables where some have been estimated using available data and others will be quantified with error simulation. The variables that are to be quantified relate to the effect that errors in the hardware have on the application results, in particular by using ResNet18 for image classification we study the decrease in accuracy as a function of an increasing number of errors.

Carbon footprint refers to the release of GHGs for in this case accelerators during various parts of their lifetimes, in this chapter we discuss embodied and operational footprint but a significant part of an accelerator's life is the end-of-life. However, this part is not covered and this project instead focuses on the footprint up until a chip reaches its end-of-life.

2.1 Carbon footprint of computers

The carbon footprint of computers encompasses a broad range of environmental impacts associated with their lifecycle, from production to disposal. This section explores key components of a computers carbon footprint, beginning with the emissions from embodied footprint tied to material extraction, manufacturing, and transport, followed by the operational footprint, which reflects emissions from energy consumed during usage. It further examines Scope 1, Scope 2, and Scope 3 emissions in section 2.1.3, providing a framework to categorize direct and indirect emissions from production and use. Given the complexity of tracking accurate emissions data, data uncertainty remains a critical challenge, impacting efforts to accurately assess and reduce emissions. Finally, this section addresses scaling trends as computing demand rises, considering how emissions evolve alongside increasing production and use.

2.1.1 Embodied footprint

The embodied footprint of computers refers to the total environmental impact that occurs throughout the entire lifetime of a computer system, from the extraction of raw materials to the disposal of the system. This includes energy consumption and resource use during manufacturing, transportation, and end-of-life disposal, as well as the waste generated in these stages. The concept contrasts with operational energy use, which accounts only for the energy consumed while the device is in use.

Computers rely on a wide range of materials, including metals like aluminum, copper, and gold, as well as Rare Earth Elements (REEs) such as neodymium, and tantalum can be found in computer systems [2]. These materials are essential for various components like circuit boards, semiconductors, and batteries. Mining these resources is energy-intensive and often involves significant environmental degradation, including habitat destruction, soil and water pollution, and high carbon emissions. Additionally, the extraction of REEs frequently produces hazardous waste that can contaminate surrounding ecosystems.

The manufacturing phase contributes significantly to a computer's embodied footprint. The process involves energy-intensive activities, including the fabrication of semiconductors, assembly of components, and production of peripherals. Manufacturing facilities often rely on energy from fossil fuels, contributing to greenhouse gas emissions. The complex supply chains involved in producing computers also increase their overall environmental impact, as materials are often shipped across the globe multiple times before the final product is assembled [7].

Once manufactured, computers must be transported from production sites to distribution centers and retailers around the world. The embodied footprint at this stage includes the carbon emissions associated with the transportation of goods via air, sea, and land. The energy used to store and package computers also adds to their environmental cost [2].

The final stage of a computers life is often problematic from an environmental standpoint. Many computers are disposed of in landfills or improperly recycled, leading to the release of toxic substances such as lead, mercury, and cadmium into the environment. Electronic waste can have a long-lasting impact, polluting water sources, soil, and air. Furthermore, the low recycling rates of critical materials like REEs mean that new extraction processes are continually required, perpetuating the environmental damage associated with raw material extraction [8].

As an example, Google's TPU v4i has an estimated embodied footprint of 386 kCO_2, the TPU v5e 402 kCO_2, and the TPU v6e 692 kCO_2 [9].

2.1.2 Operational footprint

The operational footprint of computers relates to the environmental impact associated with their use over their functional lifetime, mainly this is made up of the energy consumed and the resulting greenhouse gas emissions. While the embodied footprint accounts for environmental impacts incurred before and after use, the oper-

ational footprint encompasses the energy demands, cooling requirements, and other resources consumed during active usage. It is also likely the most significant footprint where for specialised AI-hardware it makes up 70% to 90% of total emissions [9].

Computers consume a significant amount of electricity while in operation. This electricity which is generated from a variety of sources like coal, oil, natural gas, solar, wind but still emit carbon dioxide and other greenhouse gases. High-performance computers, such as those used in data centers, are particularly energy-intensive and contribute substantially to global electricity consumption [10].

Besides consuming electricity, computers also generate considerable amounts of heat. This creates a need for cooling systems in order to maintain optimal performance and prevent overheating. Cooling systems, especially in data centers, are often energy-intensive which adds to the overall operational footprint. Large-scale facilities sometimes use water for cooling which is sometimes consumed which creates an additional demand for water which can have environmental impact when a facility is located in water scarce regions.

A computers operational footprint also includes the power needed to run connected devices, such as monitors and external storage devices, as well as networking infrastructure. Networked devices often operate continuously to maintain connectivity, consuming energy even when they are not in active use. In addition, the type and efficiency of software running on computers also affect their operational footprint. Poorly optimized software can increase a computers processing load, leading to greater energy consumption. And most importantly, the way individuals use computers can have a substantial impact on their operational footprint. Users who, fail to adjust power settings or use resource-intensive applications unnecessarily can significantly increase the energy demand. Education around sustainable practices, such as not using computer resources unnecessarily can help reduce the operational footprint.

2.1.3 Scope 1, scope 2, scope 3 emissions

Based on the GHG protocol, both Scope 1 and Scope 2 emissions contribute to an entity's overall carbon footprint, they represent different types of environmental impact. Scope 1 emissions come directly from a entitys own activities and offer more immediate feedback on emissions. Scope 2 emissions, however, are caused by an entity's action and often depend on the carbon footprint of third parties, this makes them more challenging to control or understand but they can still be influenced by long term strategies and selective cooperation with parties who have a lower carbon footprint.

The most fundamental distinction between scope 1 and scope 2 emissions lies in whether the emissions are directly or indirectly caused by an entitys activities. Scope 1 emissions are direct emissions that occur from sources controlled or owned by the entity. In the computer industry, this can include emissions from manufacturing processes that require fuel combustion, transportation of products or intermediate-

products, or emissions from facilities operated by the entity. Scope 2 Emissions, on the other hand, are indirect emissions created from an entity's activities. For example, when a computer manufacturer buys electricity to power its data centers or facilities, the emissions generated by the power plant providing that electricity are counted as scope 2 emissions or when acquiring materials for manufacturing the emissions generated by the provider of the material are also counted as scope 2 emissions.

Scope 3 emissions represent the most extensive and complex category of emissions in the carbon footprint of computers, encompassing all indirect emissions that occur across a entitys value chain, outside of its direct operations and purchased energy. These emissions originate from a variety of sources that are harder to measure and control, including upstream activities like material sourcing and downstream activities like product disposal. The biggest difference to scope 2 emissions is that scope 3 emissions also include the emissions from downstream, for example the emissions generated after a computer has been sold and is used by any number of entities with different policies and practices until its end of life.

2.1.4 Data uncertainty

Data uncertainty is a significant challenge in accurately assessing the carbon footprint of computers, affecting both embodied and operational emissions. The complexity of modern supply chains, variations in production processes, and differences in energy sources contribute to uncertainty at nearly every stage of a computers lifecycle. This uncertainty complicates efforts to calculate and compare carbon footprints since the differences can vary across types of computers and users with different requirements for manufacturing and usage.

Computers are built from many different components, each can require different raw materials, manufacturing processes, and global transportation routes. For example, a single computer may contain components sourced from multiple countries, each with distinct energy sources and emissions. Collecting accurate emissions data for each stage of material extraction, transportation, and production is challenging, and small variations can lead to substantial differences in calculated emissions. Supply chain complexity thus introduces high variability and uncertainty into embodied emissions calculations. Different manufacturing techniques and technologies further contribute to data uncertainty. Semiconductor fabrication, for example, is an energy-intensive process with emissions that can vary widely depending on the technology, age of the equipment, and the specific energy source used. Even within the same company, manufacturing emissions can differ significantly across facilities or production lines. Inconsistent practices or lack of precise emissions data at the factory level make it challenging to obtain reliable data for specific products or batches, further complicating accurate carbon footprint assessments. [2]

The operational footprint of computers heavily depends on the energy sources available in the regions where they are used and produced. Regions with high reliance on fossil fuels, for instance, can contribute more than those that draw from renewables like wind, solar, or hydroelectric power. However, companies and users in

many regions do not even have full control of the exact energy mix powering their facilities, leading to uncertainty in accurately attributing emissions. Standardized methods for tracking and reporting emissions data are still developing, especially within the rapidly changing tech industry. While frameworks like the Greenhouse Gas Protocol provide guidelines, there is often variability in how companies apply these standards. For example, indirect emissions (Scope 2 and Scope 3) are sometimes estimated based on industry averages rather than precise data from specific suppliers or energy providers, which can introduce inaccuracies. Smaller suppliers or manufacturers may also lack the resources or systems to measure their emissions accurately, forcing companies to rely on generalized assumptions. The operational carbon footprint of a computer is also influenced by the way end-users interact with their devices. User behavior such as idle time, frequency of use, and power settings vary widely, making it difficult to standardize operational emissions data. Moreover, variations in device lifespanssuch as extended use through upgrades versus early disposal and replacementfurther add to uncertainty. Tracking emissions based on assumptions about average usage patterns may not reflect real-world use, leading to either overestimations or underestimations of a computer's operational footprint [2].

The end-of-life stage introduces significant uncertainty due to differing practices in recycling, disposal, and refurbishment. Many computers end up in informal recycling sectors where emissions data is not tracked or reported accurately. Additionally, the environmental impact of e-waste varies depending on whether devices are recycled for materials, refurbished for reuse, or discarded in landfills. Inconsistent data on recycling rates, disposal methods, and emissions associated with these activities adds further uncertainty to the total carbon footprint.

2.1.5 Scaling trends

As global reliance on computing grows, scaling trends within the computer industry have significant implications for its carbon footprint. The increasing demand for devicesfrom personal electronics to industrial servershas led to a rise in production, which in turn amplifies both embodied and operational emissions. Additionally, the growth of cloud computing, artificial intelligence, and data analytics has accelerated the expansion of data centers, many of which are energy-intensive and contribute substantially to operational emissions. Efforts to improve energy efficiency in both hardware and data centers have mitigated some environmental impacts; however, overall emissions continue to increase as computing demand outpaces efficiency gains [3] [11].

2.2 Estimating carbon footprint with a simple first order model

The work of Eeckhout [4] presents a model to assess computer architecture sustainability based on first principles and in this project we expand upon it to assess the long-term sustainability with regards to the replacement of chips.

2.2.1 First order embodied footprint

The chosen proxy for embodied footprint in Eeckhout [4] is chip area. This is based on the fact that a main component of semiconductor production is the silicon wafer and with a formula from de Vries [12] the area can be used as the proxy for the entire footprint.

2.2.2 First order operational footprint

The operational footprint can be understood as the footprint caused by the operation of a chip during its lifetime and can be considered in two scenarios, fixed-work and fixed time. The fixed-work scenario is when a chip does a fixed amount of work during its lifetime, this implies that a more efficient chip could doing the same work could the operational footprint. This however does not take Jevon's paradox [13] into account, so a common scenario is the efficiency gains being offset by the user issuing more work. The fixed-time scenario takes this into account and provides a different way to characterize the operational footprint. Fixed-time assumes that the amount of time a chip is used remains constant regardless of what chip is used, with this in mind the chosen proxy for characterizing the operational footprint is the power of the chip [4].

2.2.3 Carbon footprint model

Putting the embodied footprint and the fixed-time operational footprint together we get equation 2.1 where A is the chip area, P is the chip power. However, α is chosen in $(0, 1)$ to represent the balance between the embodied and operational footprint and an appropriate value has to be chosen for different applications. This equation gives the total footprint for a chip over its lifetime and is used to include additional chip characteristics such as increasing power usage in more powerful chips or different manufacturing processes for different generations of chips.

$$F_{fixed-time} = \alpha \cdot A + (1 - \alpha)P \tag{2.1}$$

2.2.4 Carbon footprint model with replacement

To characterize the carbon footprint caused by replacement policies it is necessary to model several generations of chips. The footprint of each generation is characterized by equation 2.1 and can be separated into the embodied footprint $\alpha \cdot A$ and the operational footprint $(1 - \alpha)P$. Since the operational footprint represents the footprint over the chip's lifetime the entire footprint is divided into equal parts over its lifetime and is incurred each year. Meanwhile, the entire embodied footprint is incurred on the first year of operation of the chip and once the chip is replaced, the entire embodied footprint is incurred again. An example of this model with $\alpha = 50\%$ and the chip being replaced every 3rd year is shown in Figure 2.1.

Another aspect that the model allows us to characterize is that with increasingly complicated manufacturing processes the embodied footprint is increasing every year,

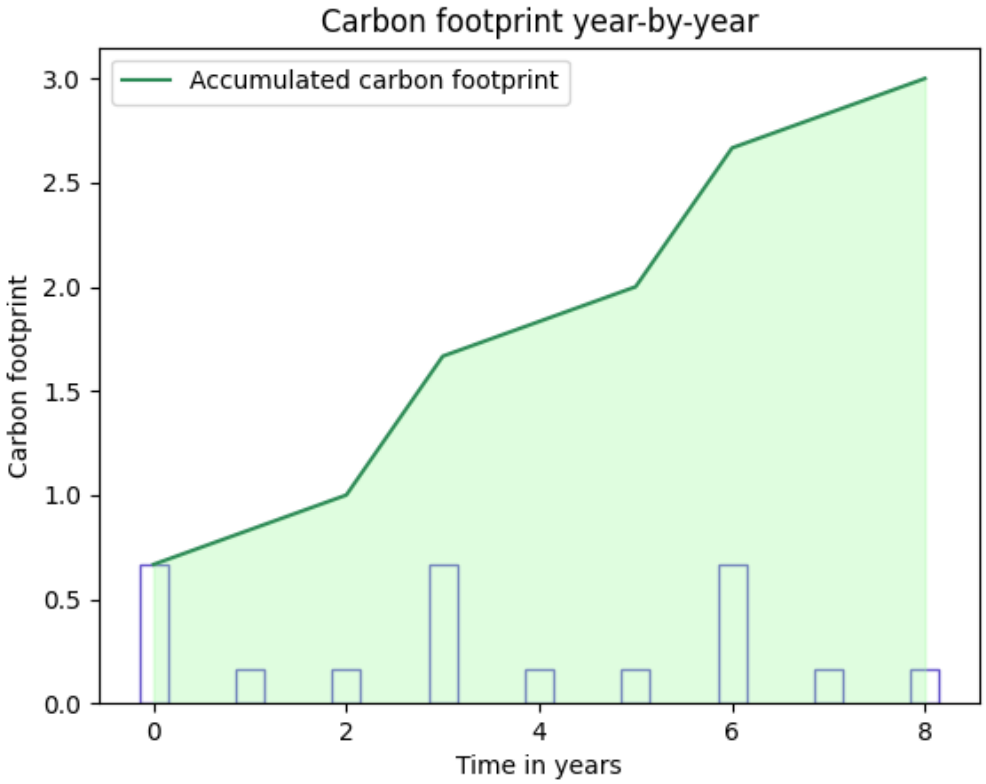


Figure 2.1: Every 3rd year the chip is replaced which incurs the embodied footprint, the operational footprint is then split over the 3 year lifespan.

with several needs of the manufacturing process expected to grow around 10% each year [14], this model assumes a 10% increase in the embodied footprint each year. Since insufficient data is available the operational footprint is assumed to be constant but the different scenarios will be discussed in section 4. The model as described with 10% increase in the operational footprint each year, $\alpha = 50\%$, and the chip being replaced every 3rd year is shown in Figure 2.2.

The final variable to be addressed is the matter of when to replace a chip, after a fixed amount of time or by some more flexible criteria. To replace a chip after a certain amount of time is intended to be analogous to replacing a chip once it reaches its end-of-life and correct execution is no longer guaranteed by the manufacturer. For this project the constant replacement time of 3 years was chosen based on the data from the Titan supercomputer [15]. Besides fixed time this project has looked into replacing a chip after a certain amount of accuracy has been lost due to errors. In this case there is a need to determine when errors start to appear $T_{time\_to\_errors}$, at what rate errors appear $T_{error\_appear\_rate}$, what impact these errors have on the accuracy $f_{error\_to\_accuracy}$, and at what loss of accuracy is the chip replaced $R_{replacement\_threshold}$. For this project, some of these variables will need to be assumed and explored since not a lot of data is available to estimate them. What will be estimated however is the effect of errors on the accuracy and this will be done through error injection.

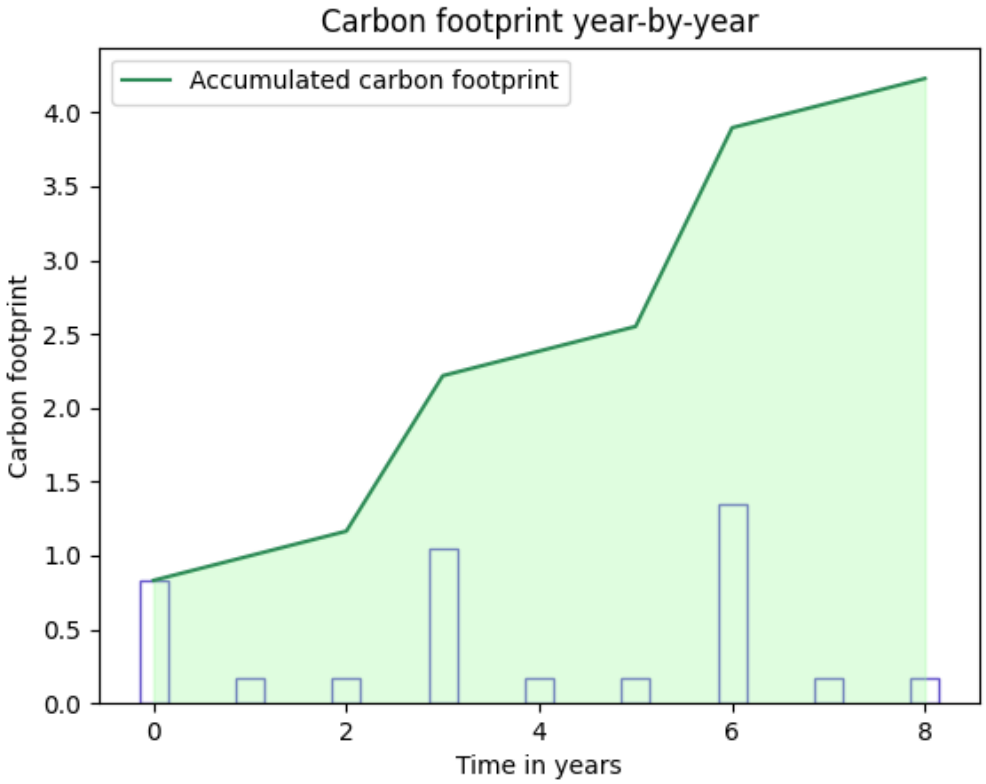


Figure 2.2: Every time the chip is replaced the increase in embodied footprint is taken into account.

Putting these considerations into the carbon footprint model we would get a slightly more complicated formula but the principle is the same. First, every time a chip is replaced the embodied footprint is incurred and like previously it will keep growing for each year. Secondly, the operational footprint that previously covered the entire lifetime of the chip is now made into the operational footprint per year $F_{op\_per\_year} = (1 - \alpha)P/lifetime\_in\_years$ and each year of operation incurs this footprint. This means that once errors start to appear it is assumed that the operational footprint remains the same. An example of this model with $T_{time\_to\_errors} = 3\ years$, $T_{error\_appear\_rate} = 10\ per\ year$, $f_{error\_to\_accuracy} = x \rightarrow x/100$, and $R_{replacement\_threshold} = 20\%$ is shown in Figure 2.3.

2.3 Errors in accelerators

In accelerators errors are considered at a very low level, at the individual bit-level. Hardware errors typically occur in the transistors whose effect typically only is a single bit and as such either a piece of data or a control signal is corrupted. These errors then propagate throughout the accelerator until they are countered by an error correction system, absorbed by the application, or affect the final result. In this project we focused on injecting errors in the software and as such are not

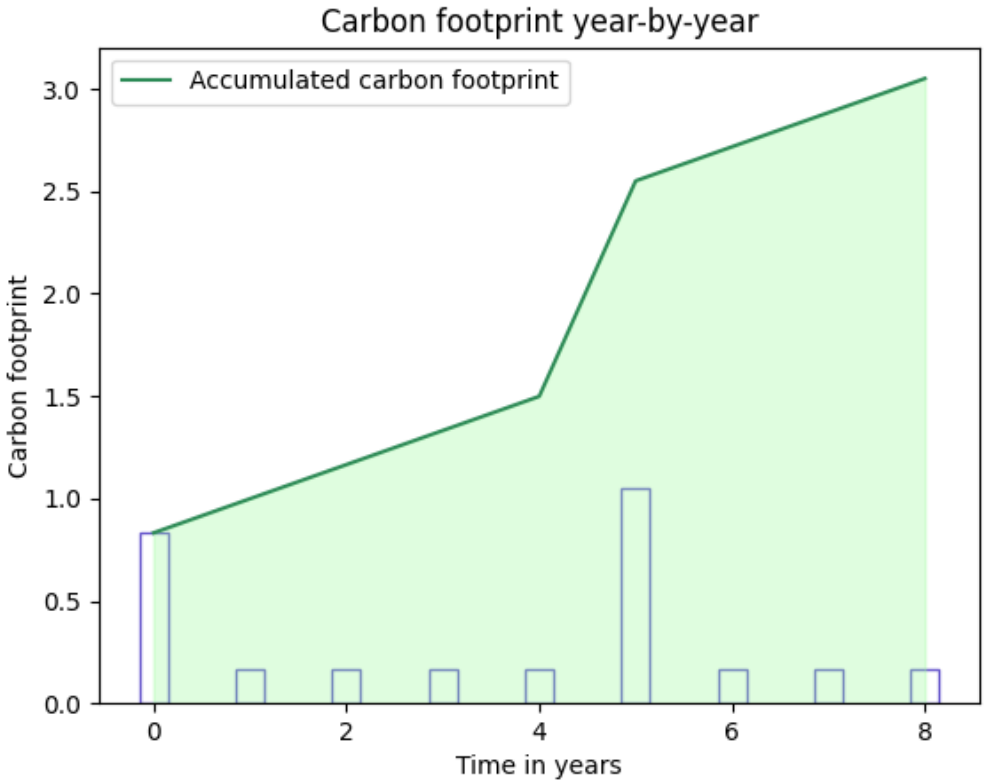


Figure 2.3: An example with an altered replacement policy which replaces its chip less often

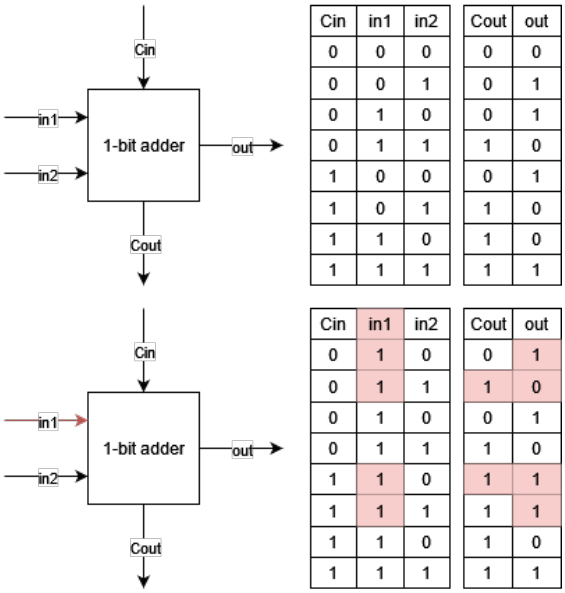
modelling any specific type of hardware error, however some suggestions for how to implement the different types of errors are provided.

While these errors can occur in any part of the accelerator in this project we are interested in the memory and the compute units. These errors occur because of several physical phenomena such as Negative-bias Temperature Instability, Hot Carrier Injection, and Electromigration. The effect of these errors can be categorized into two groups, soft errors and hard errors.

2.3.1 Hard errors

Hard errors refer to errors that always produce the same result regardless of the intended behavior. One type of hard error is the stuck-at-fault, with this error a circuit element always outputs the same value. With this type of error in an adder circuit a bit of the input could be stuck at 1 which makes some but not all outputs faulty, see Figure 2.4. Hard errors can affect the memory as well and similarly to compute elements they behave in a constant manner. This constant characteristic of hard errors makes them not too difficult to spot with unit tests and can to an extent be worked around. Given that the accelerator has the functionality supported it is possible to create a fault-map of known hard errors and their behavior, this map can then be leveraged to assign data and computations in such a way that the hard

errors effects get masked as correct behavior.



Cin	in1	in2	Cout	out
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Cin	in1	in2	Cout	out
0	1	0	0	1
0	1	1	1	0
0	1	0	0	1
0	1	1	1	0
1	1	0	1	1
1	1	1	1	1
1	1	0	1	0
1	1	1	1	1

Figure 2.4: Normal adder and a faulty adder

2.3.2 Soft errors

Soft errors refer to errors that only produce the wrong result some of the time. An example of this is timing errors, with this error the circuit has degraded and no longer works at the speed it was designed to operate at. As a result, when the next clock cycle starts it is possible that a circuit has not reached its steady state and an erroneous value is propagated. This uncertainty with soft errors can make them more difficult to spot as unlike hard errors they do not have the same constant characteristic, this means that unit tests may not be enough to detect these soft errors and an online error detection system may have to be used instead. To counter soft errors it is possible to do the same as with hard errors and create a fault-map but since soft errors do not necessarily produce the wrong results it can be a bit much to introduce a fault-map. Another common countermeasure is to simply lower the operating frequency to lower the timing requirements on the circuit.

2.3.3 Error mitigation

In modern computing there a variety of methods used to mitigate these errors such as workload-aware redundancy which focuses on assigning redundant computational resources based on the criticality and sensitivity of workloads. Instead of applying uniform redundancy across all tasks, systems can selectively replicate or verify computations for high-priority or error-sensitive workloads while relaxing redundancy for tolerant ones, such as AI inference tasks. This targeted approach minimizes the performance and energy costs associated with redundancy while maintaining system reliability where it matters most [16].

Another method is fleet-wide error correlation which leverages large-scale monitoring and data aggregation across large numbers of accelerators to detect patterns of degradation and failure. By correlating error occurrences across similar hardware units, systems can predict which components are likely to experience hard errors before they manifest critically. This enables proactive mitigationsuch as reassigning workloads, adjusting operating parameters, or scheduling maintenancebefore the error occurs [16].

Finally, Task Distribution serves as a complementary strategy by intelligently routing workloads away from unreliable or degraded hardware. Rather than immediately decommissioning aging accelerators, systems can assign them less critical or error-tolerant tasks, maximizing resource utilization while minimizing the risk of computation failure. Combined, these strategies can create a resilient computing system in which the impact of hard errors is minimized and hardware lifespans are extended through various methods [17] [18].

3

Method

3.1 Implementing the carbon footprint model

For this project we implemented the model as a python script that estimated the carbon footprint over 20 years and compared between switching the chip after 3 years and a certain loss of application accuracy. Time is counted discreetly year for year so each year the model checks how long the chip has been in use and depending on the predicted reduction in accuracy and current replacement policy and may replace the chip if accuracy has dropped below acceptable levels. At each timestep the operational footprint is added to the carbon footprint and if the chip is replace a unit of the embodied carbon footprint, the model also takes into account the increasing footprint of such chips over time.

In order to ascertain the loss of application accuracy over time it is necessary to know how any number of errors affect the application and the result on application accuracy. In this project we decided to assume the number of errors affecting the application instead of figuring out at what rate errors accumulate and instead focus on a method for determining what effect a certain number of errors have on application accuracy, how this was implemented is explained in the section below.

3.2 Error injection

A brief explanation and the algorithms used to inject errors are available in Appendix A.

3.2.1 Error injection in software

Injection errors in software is the simplest option to implement but requires the most knowledge of error characteristics to be realistic. Similar to real errors we can choose to implement the injection at the compute units or in the memory, regardless of where it is prudent to modify a correctly functioning program to add the error injection functionality. To inject errors into the memory an alternative is to pause the execution, modify the variables, and then continue the execution. The functionality required for this is present in debugging software such as GDB [19]. When modifying the memory we can use a fault-map to always modify the same memory locations with either a hard error by modifying it to a consistent value or

if it is a soft error we can do so randomly. On the other hand, to inject errors in the compute units we could modify the code itself so that the calculations are erroneous on purpose, similarly as with memory we can use a fault-map and characterize the errors similarly in order to mimic hard and soft errors.

3.2.2 Error injection in hardware

Injecting errors in hardware can be very dependent on which hardware architecture is targeted since it is up to the vendor to provide sufficient APIs or other interfaces to the hardware. An example of this is NVBitFI [20] which is an error injection tool based on NVBit [21] which allows users to modify the GPU's assembly code. NVBitFI has quite strict requirements for what architectures are compatible along with what compilers can be used. This makes for a rather tricky development environment since applications also have to be compiled and linked with specific versions. And this is not limited to NVBitFI, its successor SASSIFI [22] also has the same strict requirements for architectures and compiler which lead to error injection in hardware being unavailable due to lack of compatible hardware.

3.2.3 Error injection with PyTorch

The chosen tool for error injection in this project ended up being the tool PyTorchFI [23], the tool is integrated with the popular deep learning platform PyTorch which makes it easy to use and integrate into existing python scripts. PyTorchFI uses hooks which is a tool provided by PyTorch which allows PyTorchFI to modify both neurons and weights in a lightweight manner, it also allows an easy way to write custom functions which can be leveraged to model different types of errors. The function we implemented only targets a single layer at a time and when doing so injects errors into random neurons each time selected by python's pseudo random number generator random.py, the injected error each time was a random bitflip where the chosen bit also was selected by random.py. This function most closely models hard errors since errors are always encountered compared to soft errors which may or may not affect a calculation.

4

Results

4.1 Experiment setup

The chosen dataset for this project was the ILSVRC2012 [6], in particular it was the validation set along with its solution. Due to restrictions in computing power only the first 1000 images were used for error injection with PyTorchFI. All the code along with results is available at https://git.chalmers.se/bjornfo/msc_thesis and a description of project workflow is available below.

4.1.1 AlexNet

Firstly, AlexNet was run with altered images where a number of bitflips had been introduced into the images to simulate errors, see workflow in Figure 4.1. The scripts modify_images.py and modify_images_parallel.py were used to randomly insert a specified number of bitflips into the images, both use the same method to flip bits but the parallel version was created to speed up execution for additional bitflips.

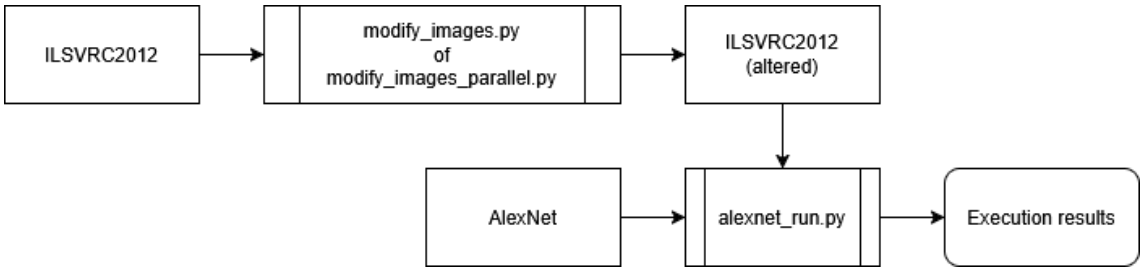


Figure 4.1: AlexNet with errors inserted into the images

Secondly, Alexnet was also run with an error injection framework using PyTorchFI. This allowed us to insert a custom function onto layers to run during inference which simulated a specified number of bitflips, see workflow in Figure 4.2.

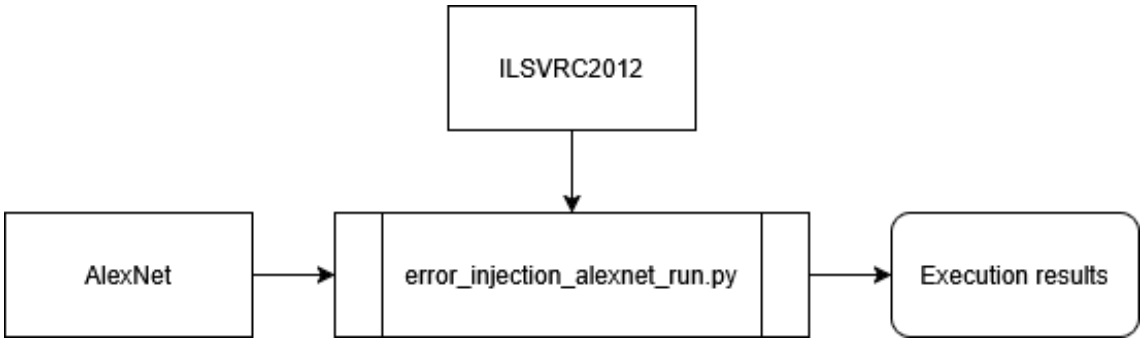


Figure 4.2: AlexNet with errors inserted during inference

4.1.2 ResNet18

For ResNet18 we used the same PyTorchFI framework as for AlexNet but parallelised it in order to speed up execution, see workflow in Figure 4.3. The improved execution speed was necessary in order to test the effect of an increasing amount of errors for every layer of ResNet18.

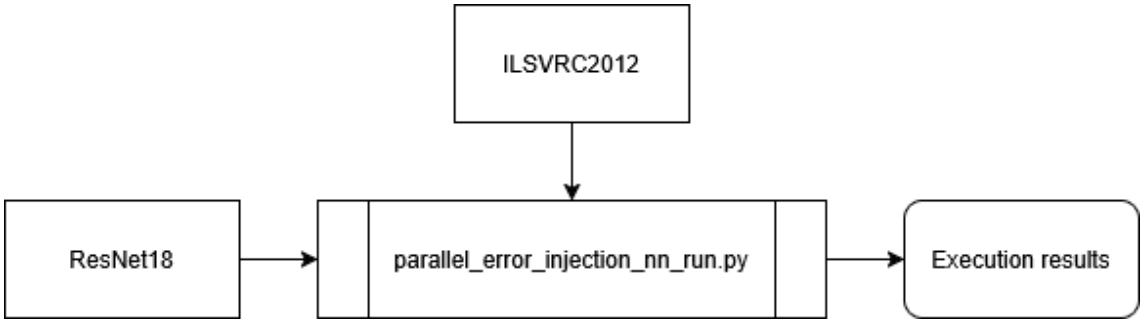


Figure 4.3: ResNet18 with errors inserted during inference

4.1.3 Results and graphs

The execution results for each scenario were compared against the solution and the accuracy for each layer was calculated and then graphed, see workflow in Figure 4.4. The execution results were in form of the filename followed by five tags corresponding to the five best guesses for what the image contained, the script stats_run.py compared the guesses to the solution and calculated the error rate for allowing the 1-5 best guesses and outputs the error rates for graphing.

The scripts draw_graph_alexnet.py and draw_graph_resnet.py both compare the degraded error rates to the error rate without any fault injection and then draws a graph using Matplotlib. Then the script carbon_model.py implements the carbon footprint model as described in section 2.2 using the calculated error rates and some user parameters to draw the graphs shown below, as an example see Figure 4.8

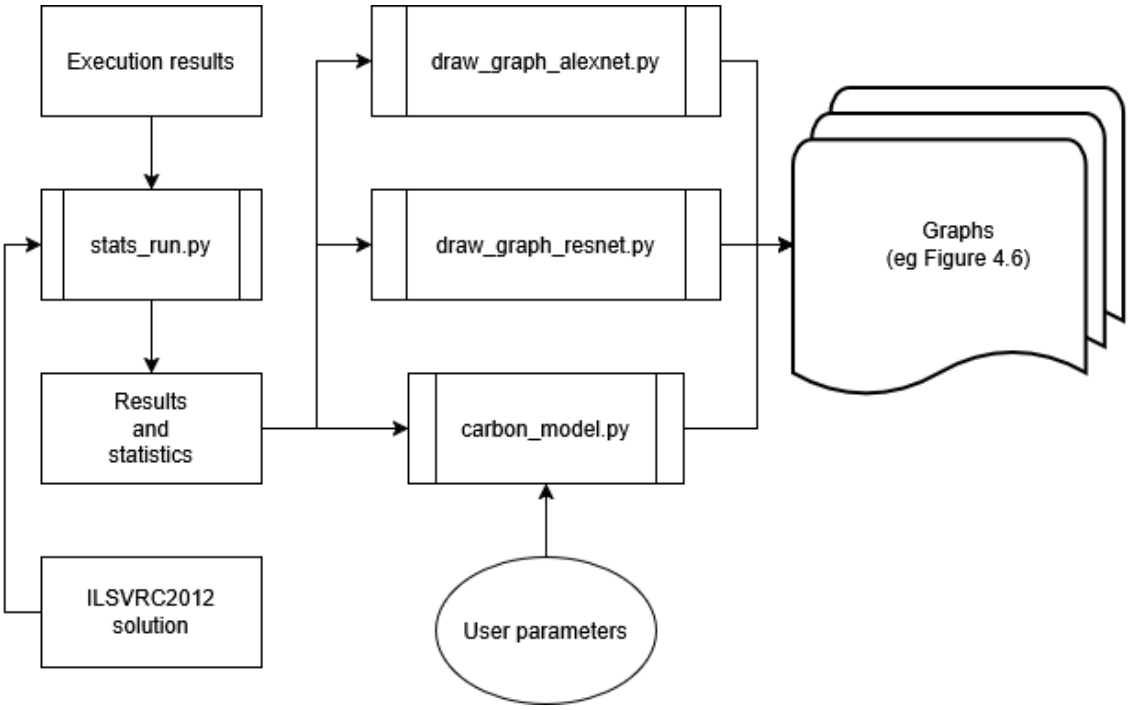


Figure 4.4: Execution results are interpreted and the statistics get turned into graphs

4.2 Fault injection in ResNet18's convolutional layers

In Figure 4.5 we see the increase in error rate from injecting errors into a layer, the increase is shown in percentage starting from its unaffected error rate. The figure shows the effect for top 1 and top 5 scenarios which start out with error rates $30,24\%$ and $10,92\%$ respectively. The horizontal red line shows where the application has an error rate of 50% which corresponds to the application having a 50% accuracy in classifying the images.

Perhaps unsurprisingly we can see that increasing the number of errors injected significantly increases the error rate which corresponds to a decrease in the applications classification accuracy. Only one layer in the top 5 case manages to retain an error rate below 50% at 100 errors.

4.2.1 Converting errors to loss in accuracy

In Figure 4.6 we can see the curves from averaging the results in Figure 4.5 and then using numpy.polyfit() function to fit the curve to a n-th degree polynomial, the degree is chosen with regards to the data points from 4.5. The graph is limited to 100 errors since an error rate above 50% is beyond reasonable for any application modelled in the carbon footprint model. The curve can now be referenced by the carbon footprint model to approximate the loss of accuracy given a certain number of errors.

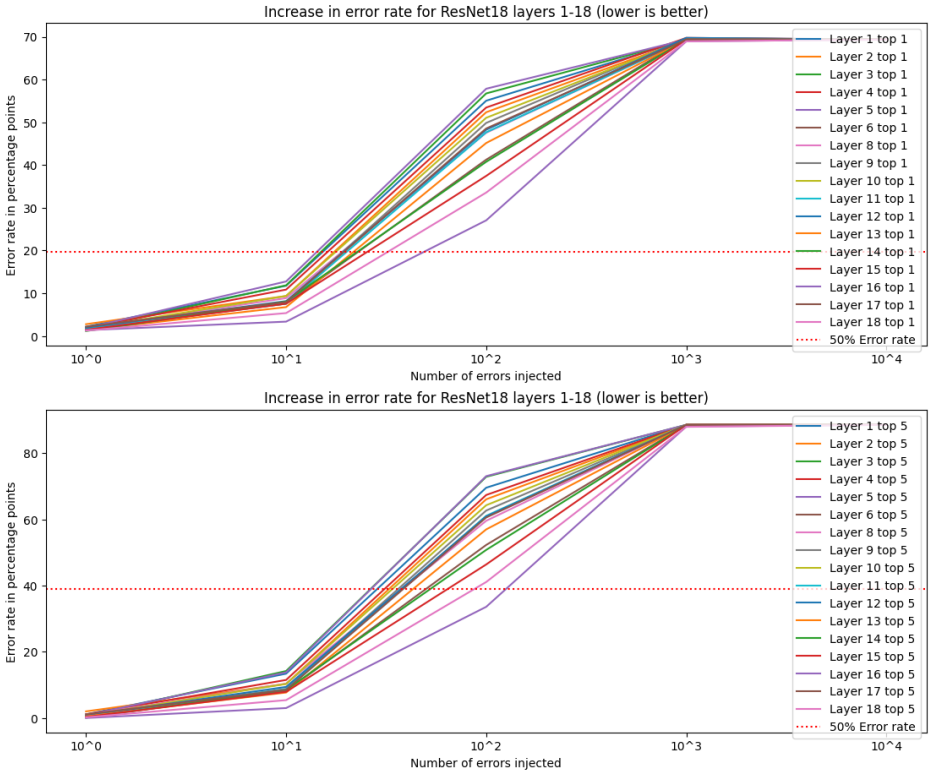


Figure 4.5: Increase in error rate for ResNet18

In this section, figures such as 4.7 will be used to compare and contrast different aspects and implications of the carbon footprint model described in section 2.2.4. The top graph plots two scenarios, replace after a fixed amount of time and replace once a certain increase in error rate has been reached, the bars indicate the footprint each year and the lines indicate the total footprint up until that year. The second graph shows the difference between these two scenarios which indicates if the altered replacement policy gives a larger or smaller carbon footprint. Unless otherwise specified the parameters will be set at $\alpha = 50\%$ and $fault\_tolerance = 20\%$, also do note that scale of the y-axis across different graphs is not consistent.

4.3 Effect of model parameters

In this section we discuss two model parameters, the α value and increasing footprint over time, although these are present in the original model as presented in 2.2.4 here we discuss their interaction in the context of a DNN application.

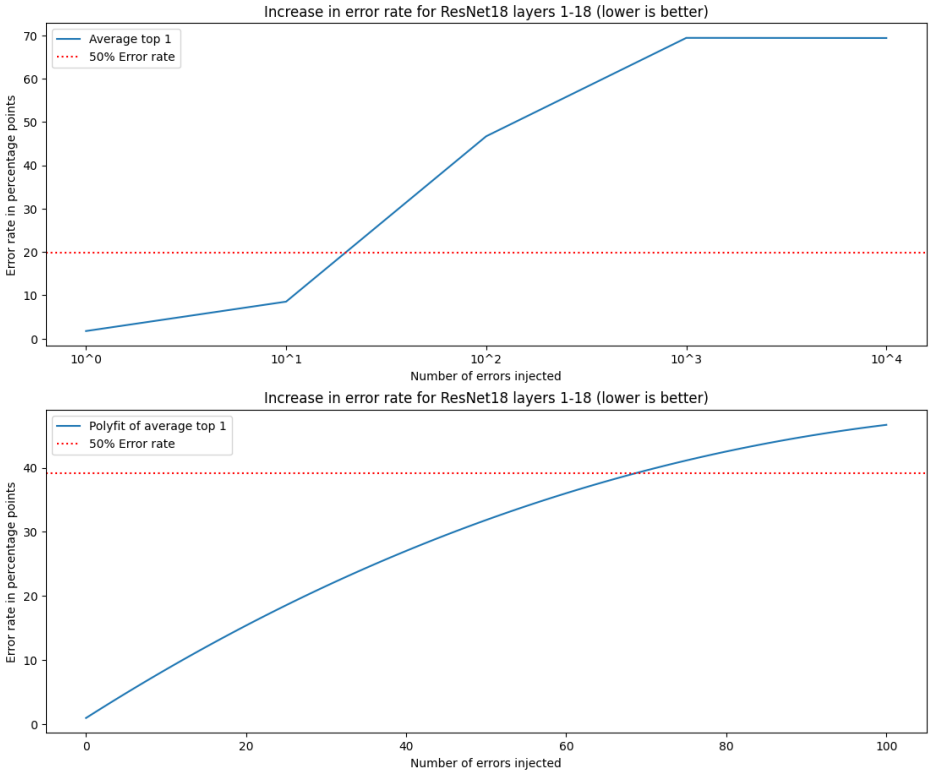


Figure 4.6: Approximated error to accuracy curve

4.3.1 Balance between embodied and operational footprint

One of the strengths of the model proposed by Eeckhout [4] is that by varying the α value we can reason about the balance between embodied and operational footprint and what implications this has for a proposed chip design. As a reminder, low α value means that the operational footprint is more prominent and that a high α value means that the embodied footprint is more prominent.

$$F_{fixed-time} = \alpha \cdot A + (1 - \alpha)P$$

The low α scenario is presented in Figure 4.8 with α values $10\%, 20\%$, and 30%. This scenario demonstrates an increase in carbon footprint with a lowered α value.

The low α scenario is presented in Figure 4.9 with α values $70\%, 80\%$, and 90%. This scenario demonstrates an decrease in carbon footprint with a higher α value.

In Figure 4.10 we see a wide representation of different α values and a strong correlation with higher α values giving a lowered carbon footprint. Since the proposed alternative is one with a different replacement policy it is reasonable that the more prominent the embodied footprint is the more impact we can have with a lowered

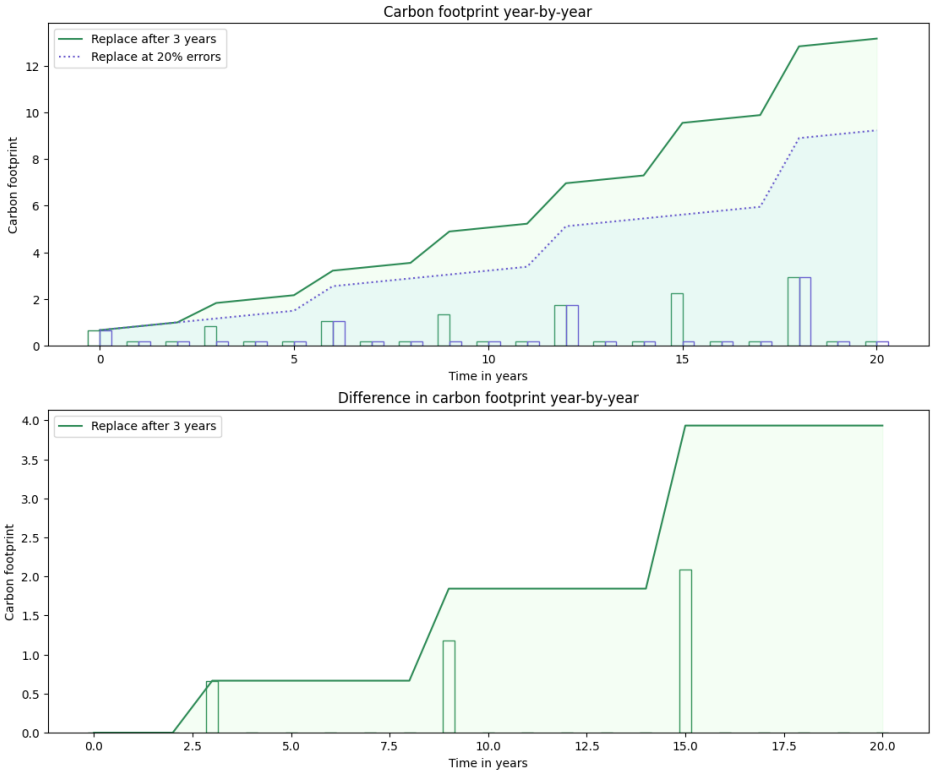


Figure 4.7: Carbon model simulation example

rate of replacement.

4.3.2 Increased operational footprint with faulty hardware

The model assumes that the embodied footprint increases over time but that the operational footprint remains constant, here we explore the effects of an increased operational footprint when using hardware with faults. The idea is that the application needs to be rerun to get the correct result or the faulty hardware consumes work resources to operate such as increased power need or increased need for cooling. In model this is implemented such that when a chip reaches its end of life and starts accumulating errors it also increases its operational footprint, in Figure 4.11 we see scenarios with $100\%, 300\%$, and 500%.

The 100% scenario has the same parameters as the example 4.7 as can be seen as a baseline in this case. For the 300% scenario we can see that for some years, such as year 4 and year 5, the altered policy has a higher footprint due to the increased operational footprint, but still the accumulated footprint is never lower than for the fixed replacement policy. However, with the 500% operational footprint there is quite some time where the altered replacement policy performs worse than the fixed replacement and even if it performs better some years and goes into the positive on

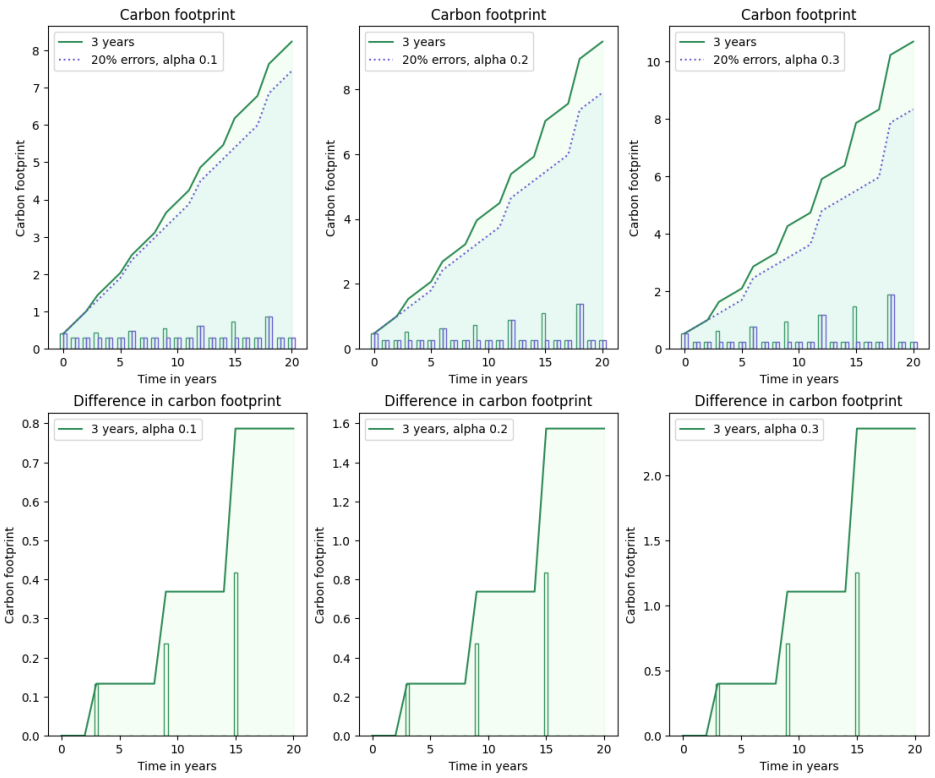


Figure 4.8: Low α model

years that the fixed policy replaces the chip, the majority of the time is spent in the negative. On the other hand, it seems that the increased operational footprint is being offset by the increasing embodied footprint and that by the end of the simulation it has almost caught up.

The reason that the increased operational footprint gets amortized over time is likely because the embodied footprint rises by 10% each year while the operational footprint remains the same. If the operational footprint were to rise each year similar to the embodied footprint it is likely that even the 300% scenario could put the altered replacement policy in the negative. It would create a scenario where the embodied footprint for acquiring a new chip would have to be higher than the increased operational footprint due to faulty hardware and the accumulated increased embodied footprint that would come from acquiring a new chip a few years later.

4.4 Replacement parameters

In this project we have proposed a simple policy for chip replacement and in this section we discuss two aspects of how that policy is implemented.

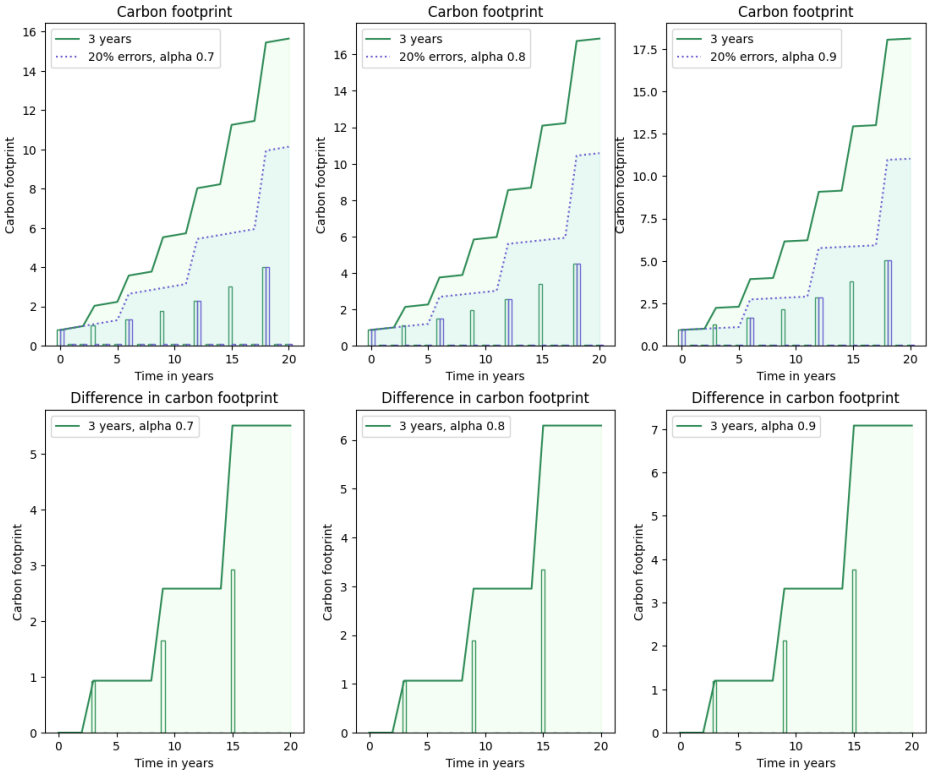


Figure 4.9: High α model

4.4.1 Different levels of fault tolerance

Depending on the use case of the DNN a different fault tolerance may be accepted and in Figure 4.12 we explore three scenarios 15%, 10%, and 5% fault tolerance.

In this case it seems that the scenario for 15% and 10% end up being identical, likely because the timesteps are large enough that both scenarios have the same outcome, a smaller timestep would likely bring different results. For the 5% scenario however we see a difference in that the difference in footprint is lower at its peak which is reasonable since the chip is being replaced more often.

An interesting thing to note however is that in this simulation all scenarios end with a sharp decrease in the carbon footprint difference, this is because all scenarios end up incurring the embodied footprint in the final year however the implication is interesting. By using the chip for longer we also acquire a new chip later, acquisition of a chip whose embodied footprint has risen by 10% each year. If the increase in embodied footprint was not linear and instead increased different amounts from year to year it could even be that by waiting, we are unlucky and end up with an even higher embodied footprint. However, the suggested replacement policy allows for the alternative to acquire a new chip at the 3-year mark in the trend is favorable and then keep using the old chip until it hits the tolerance level until doing the actual

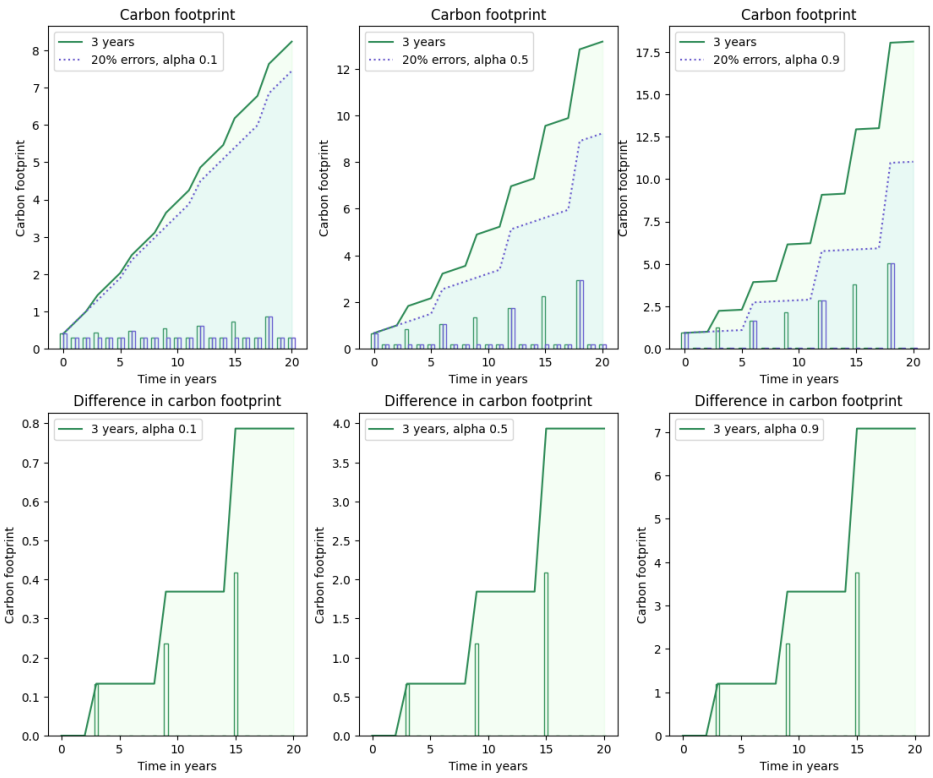


Figure 4.10: Varied α model

chip replacement, on the other hand if at 3 years the trend is unfavorable then we can wait until the tolerance level is hit to acquire the new chip. In other words, the proposed replacement policy allows for some flexibility to adapt to the embodied footprint trend.

Also for the 5% it is negative during two years, year 4 and year 8, as well as ending the simulation very close to 0, an implication from this is that the savings done by not acquiring a new chip can be negated by the increased embodied footprint due to time.

4.5 Summary

In summary it is possible to lower the footprint somewhat by using an altered replacement policy where some reduction in model accuracy is tolerated. However, in the scenarios where faulty hardware has a higher operational footprint the fault tolerance only has a lower footprint in the long run. Additionally, when the fault tolerance is varied we see an effect where by using hardware only for a little longer before replacement, the trend of increasing embodied footprint may cause the fault tolerant policy to incur a higher footprint upon replacement due to increased em-

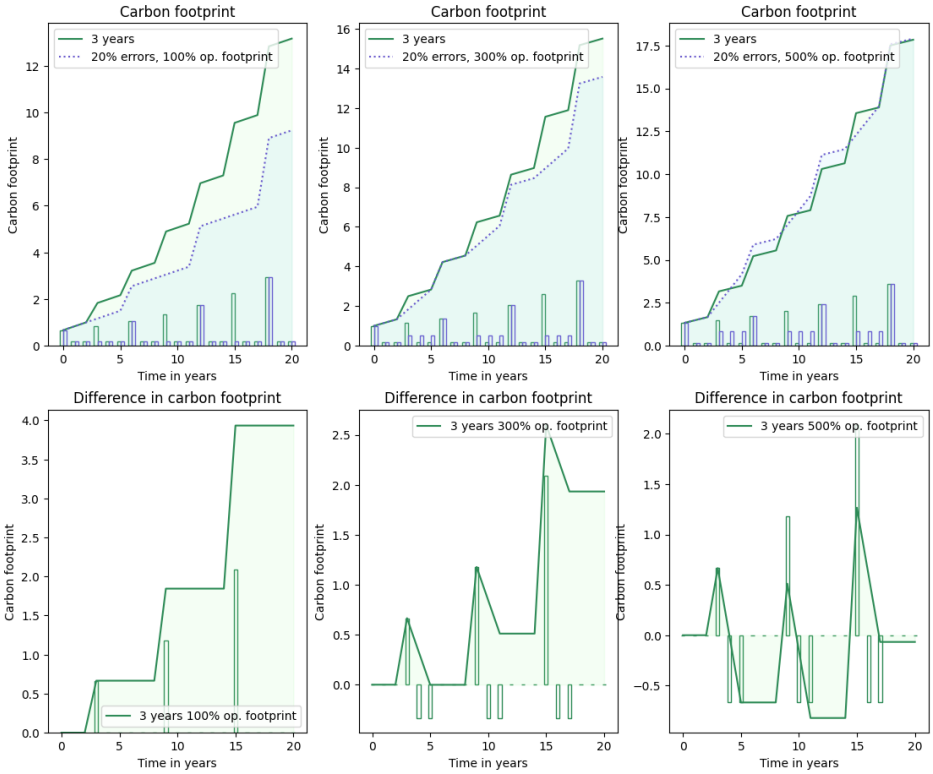


Figure 4.11: Increased operational footprint model

bodied footprint.

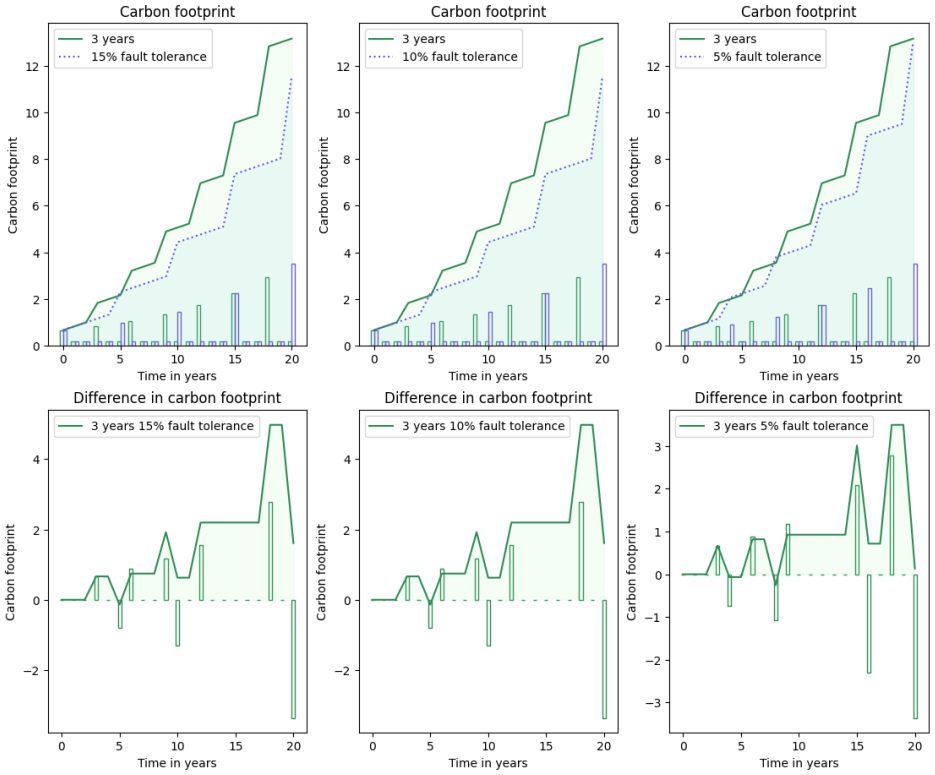


Figure 4.12: Varied fault tolerance level

5

Conclusions

5.1 Accelerators towards end of life

In conclusion, while accelerators tend to show an increased rate of errors as they approach the end of their lifecycle, they can still offer a positive impact when carefully managed. By balancing the replacement rate with performance requirements, organizations can extend the useful life of these accelerators while maintaining acceptable output. This strategy allows for a gradual replacement approach, optimizing resources and cost while achieving the desired computational performance. Proper planning and maintenance can thus ensure that aging accelerators continue to contribute effectively, minimizing environmental and financial impacts associated with frequent replacements.

Certain AI applications, particularly those involving tasks like image recognition, can tolerate moderate levels of computational errors without significant degradation in performance. Many machine learning algorithms are designed to be resilient to minor inaccuracies due to their probabilistic nature. That is, errors in individual computations may introduce noise but often dont prevent the model from producing useful results. This tolerance allows older hardware, even with a higher error rate, to remain viable in such applications.

However, as error rates increase beyond a certain threshold, they begin to disrupt the consistency and reliability of model outputs, ultimately deteriorating performance to a point where the hardware is no longer useful. High error rates can lead to incorrect predictions, decreased accuracy, and misclassification in AI tasks, affecting the overall quality and trustworthiness of the application. For this reason, even with error-tolerant AI workloads, it is essential to monitor error rates and establish replacement policies that keep performance within acceptable limits while maximizing the hardwares lifecycle.

Allowing a controlled level of error in AI applications can yield substantial long-term advantages by fostering robustness and adaptability in model development. This approach also offers sustainable benefits, as it reduces the frequency of hardware replacements, allowing resources to be redirected toward refining and optimizing algorithms rather than continuous hardware upgrades. In the long term, this can cultivate a more sustainable and efficient technology ecosystem, where AI models become progressively more resilient and adaptable to real-world variability and hard-

ware inconsistencies. By planning for and accepting manageable errors, the industry can enhance the durability and robustness of AI, ensuring its capability to evolve alongside and capitalize on hardware improvements over time.

5.2 Limitations encountered

The biggest limit encountered was the available tools for error injection. All of the explored hardware injectors were for GPUs of unavailable architectures or different manufacturers than what was available or in the case that an injector was available it was for significantly older software. One way to solve this would be to search for an available GPU compatible with current hardware injectors at a larger scale than this project which put little energy into finding new hardware besides immediately available institutional resources and personal hardware. Another solution would have been to set up a dedicated lab machine with an older suite of software alongside a rewrite of some project code to be compatible. Another limitation was the small scope of models, datasets, and error characteristics considered which put limits on the project with regards to exploring more alternatives. For example the ability to vary the predictive model could provide more insights into the inherent reliability of DNNs, varying datasets would have given better results as the guidelines and principles for assembling datasets has evolved since ILSVRC2012 [6] was created. And finally, due to the difficulty of implementation very few types of errors were ever implemented which restricted the insights gained into the error resilience of DNNs.

5.3 Further work

5.3.1 Long-term studies

Long-term studies on aging accelerators offer valuable insights into the lifecycle and reliability of these critical computing components, with the potential to influence both hardware design and operational strategies. By analyzing how accelerators perform as they approach the end of their lifecycle, we can uncover trends in error rates, energy efficiency, and computational output. These insights are essential for developing predictive models that help identify when performance degradation reaches a point where replacement or intervention is necessary.

During the background reading for this paper, we found few sources where efforts to keep a significant number of accelerators running for a long time. The benefits of such an effort is to provide more data regarding the lifetime of accelerators, where accelerators are most likely to break first, the effect of different loads on accelerators lifetime, and the effect of degraded accelerators on tasks. These are questions that are difficult to answer with current tools as they either require non-existent data or sophisticated techniques and access to hardware schematics.

5.3.2 Better methods for error mitigation

One aspect not explored in this paper is the potential effect of error mitigation techniques on AI accelerators. Since AI applications have some resilience towards errors it would be interesting to know if error mitigation techniques have any impact at all, if it extends the existing resilience of AI applications, or if it provides little effect at all.

5.3.3 Better understanding of user attitude towards faulty hardware

It is generally understood that potential errors in any application is unwanted and if possible shall be removed. However, with AI applications it is necessary to accept some risk that the result is incorrect, therefore it would be interesting to know if users would be willing to accept a reduction in correctness for improvements elsewhere.

Bibliography

[1] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Lincoln ai computing survey (laics) update," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, 2023, pp. 1–7. DOI: 10.1109/HPEC58863.2023.10363568.

[2] K. Kirkpatrick, "The carbon footprint of artificial intelligence," *Commun. ACM*, vol. 66, no. 8, pp. 17–19, Jul. 2023, ISSN: 0001-0782. DOI: 10.1145/3603746. [Online]. Available: https://doi.org/10.1145/3603746.

[3] C. Freitag, M. Berners-Lee, K. Widdicks, B. Knowles, G. S. Blair, and A. Friday, "The real climate and transformative impact of ict: A critique of estimates, trends, and regulations," *Patterns (New York, N.Y.)*, vol. 2, no. 9, p. 100 340, Sep. 2021, ISSN: 2666-3899. DOI: 10.1016/j.patter.2021.100340. [Online]. Available: https://europepmc.org/articles/PMC8441580.

[4] L. Eeckhout, "A first-order model to assess computer architecture sustainability," *IEEE Computer Architecture Letters*, vol. 21, no. 2, pp. 137–140, 2022. DOI: 10.1109/LCA.2022.3217366.

[5] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17 322–17 341, 2017. DOI: 10.1109/ACCESS.2017.2742698.

[6] O. Russakovsky, J. Deng, H. Su, *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[7] M. Ruberti, "The chip manufacturing industry: Environmental impacts and eco-efficiency analysis," *Science of The Total Environment*, vol. 858, p. 159 873, 2023, ISSN: 0048-9697. DOI: https://doi.org/10.1016/j.scitotenv.2022.159873. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S004896972206973X.

[8] K. Liu, Q. Tan, J. Yu, and M. Wang, "A global perspective on e-waste recycling," *Circular Economy*, vol. 2, no. 1, p. 100 028, 2023, ISSN: 2773-1677. DOI: https://doi.org/10.1016/j.cec.2023.100028. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2773167723000055.

[9] I. Schneider, H. Xu, S. Benecke, *et al.*, *Life-cycle emissions of ai hardware: A cradle-to-grave approach and generational trends*, 2025. arXiv: 2502.01671 [cs.AR]. [Online]. Available: https://arxiv.org/abs/2502.01671.

[10] J. Malmodin, N. Lövehagen, P. Bergmark, and D. Lundén, "Ict sector electricity consumption and greenhouse gas emissions 2020 outcome," *Telecommunications Policy*, vol. 48, no. 3, p. 102 701, 2024, ISSN: 0308-5961. DOI: https:

//doi.org/10.1016/j.telpol.2023.102701. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0308596123002124.

[11] C.-J. Wu, R. Raghavendra, U. Gupta, *et al.*, "Sustainable ai: Environmental implications, challenges and opportunities," in *Proceedings of Machine Learning and Systems*, D. Marculescu, Y. Chi, and C. Wu, Eds., vol. 4, 2022, pp. 795–813. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2022/file/462211f67c7d858f663355eff93b745e-Paper.pdf.

[12] D. de Vries, "Investigation of gross die per wafer formulas," *IEEE Transactions on Semiconductor Manufacturing*, vol. 18, no. 1, pp. 136–139, 2005. DOI: 10.1109/TSM.2004.836656.

[13] B. Alcott, "Jevons' paradox," *Ecological Economics*, vol. 54, no. 1, pp. 9–21, 2005, ISSN: 0921-8009. DOI: https://doi.org/10.1016/j.ecolecon.2005.03.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921800905001084.

[14] M. Garcia Bardon, P. Wuytens, L.-Å. Ragnarsson, *et al.*, "Dtco including sustainability: Power-performance-area-cost-environmental score (ppace) analysis for logic technologies," in *2020 IEEE International Electron Devices Meeting (IEDM)*, 2020, pp. 41.4.1–41.4.4. DOI: 10.1109/IEDM13553.2020.9372004.

[15] G. Ostrouchov, D. Maxwell, R. A. Ashraf, C. Engelmann, M. Shankar, and J. H. Rogers, "Gpu lifetimes on titan supercomputer: Survival analysis and reliability," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–14. DOI: 10.1109/SC41405.2020.00045.

[16] S. Gupta, "Reducing hardware-related interruptions in ai clusters: Strategies for resilient gpu infrastructure," English, *Journal of International Crisis and Risk Communication Research*, vol. 8, pp. 44–53, 2025. [Online]. Available: http://proxy.lib.chalmers.se/login?url=https://www.proquest.com/scholarly-journals/reducing-hardware-related-interruptions-ai/docview/3257197972/se-2.

[17] R. Boëzennec, F. Dufossé, G. Pallez, and A. Tremodeux, "Improving Supercomputer Usage with Aging Awareness," in *Sustainable Supercomputing (Workshop of SC25)*, St. Louis, Missouri, United States, Nov. 2025. [Online]. Available: https://hal.science/hal-05109521.

[18] T. B. Hewage, S. Ilager, M. R. Read, and R. Buyya, "Aging-aware cpu core management for embodied carbon amortization in cloud llm inference," in *Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems*, ser. E-Energy '25, New York, NY, USA: Association for Computing Machinery, 2025, pp. 43–55, ISBN: 9798400711251. DOI: 10.1145/3679240.3734608. [Online]. Available: https://doi.org/10.1145/3679240.3734608.

[19] *GDB: The GNU Project Debugger — sourceware.org*, https://sourceware.org/gdb/, [Accessed 15-04-2024].

[20] T. Tsai, S. K. S. Hari, M. Sullivan, O. Villa, and S. W. Keckler, "Nvbitfi: Dynamic fault injection for gpus," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 284–291. DOI: 10.1109/DSN48987.2021.00041.

[21] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "Nvbit: A dynamic binary instrumentation framework for nvidia gpus," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52, Columbus, OH, USA: Association for Computing Machinery, 2019, pp. 372–383, ISBN: 9781450369381. DOI: 10.1145/3352460.3358307. [Online]. Available: https://doi.org/10.1145/3352460.3358307.

[22] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 249–258. DOI: 10.1109/ISPASS.2017.7975296.

[23] A. Mahmoud, N. Aggarwal, A. Nobbe, *et al.*, "Pytorchfi: A runtime perturbation tool for dnns," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020, pp. 25–31.

A

Algorithms used

```
1 for i in flip_list:
2         bim[i] = (bim[i] + random.randrange(256)) % 256
```

Listing A.1: Algorithm for flipping bits in an image represented as a byte array

The image to be corrupted is turned into a byte array *bim* which at the same time converts all images to an RGB format. Then a list *flip_list* is generated which contains all the indices in the byte array of where to flip a bit, this list is sorted such that the byte array is accessed in order. Finally a byte is accessed at index i and a bit-flip is simulated with the formula on row 2.

```
1 fs = pack('f', output[0, x, y, z].item())
2 bval = list(unpack('BBBB', fs))
3 [q,r] = divmod(random.randrange(32), 8)
4 bval[q] ^= 1 << r
5 fs = pack('BBBB', *bval)
6 fnew=unpack('f', fs)
7 a = fnew[0]
```

Listing A.2: Algorithm for flipping a bit in a python 32-bit float taken from `https://stackoverflow.com/a/34679225`