



# Master thesis

Degree of Master of Science in Engineering,  
Computer Science and Engineering, 300  
credits

## Attacks and Vulnerabilities of Hardware Accelerators for Machine Learning: Degrading Accuracy Over Time by Hardware Trojans

Computer Science and Engineering, 30 credits

Halmstad 2024-07-01  
Marcus Niklasson & Simon Uddberg



Marcus Niklasson & Simon Uddberg: *Attacks and Vulnerabilities of Hardware Accelerators for Machine Learning: Degrading Accuracy Over Time by Hardware Trojans*, , © July 1, 2024

UNIVERSITY:

Halmstad University

LOCATION:

Halmstad

SUPERVISORS:

Mahdi Fazeli

Ahmad Patooghy

TIME FRAME:

November 2023 - June 2024





## ABSTRACT

---

The increasing application of Neural Networks (NNs) in various fields has heightened the demand for specialized hardware to enhance performance and efficiency. Field-Programmable Gate Arrays (FPGAs) have emerged as a popular choice for implementing NN accelerators due to their flexibility, high performance, and ability to be customized for specific NN architectures. However, the trend of outsourcing Integrated Circuit (IC) design to third parties has introduced new security vulnerabilities, particularly in the form of Hardware Trojans (HTs). These malicious alterations can severely compromise the integrity and functionality of NN accelerators.

Building upon this, this study investigates a novel type of HT that degrades the accuracy of Convolutional Neural Network (CNN) accelerators over time. Two variants of the attack are presented: Gradually Degrading Accuracy Trojan (GDAT) and Suddenly Degrading Accuracy Trojan (SDAT), implemented in various components of the CNN accelerator. The approach presented leverages a sensitivity analysis to identify the most impactful targets for the trojan and evaluates the attack's effectiveness based on stealthiness, hardware overhead, and impact on accuracy.

The overhead of the attacks was found to be competitive when compared to other trojans, and has the potential to undermine trust and cause economic damages if deployed. Out of the components targeted, the memory component for the feature maps was identified as the most vulnerable to this attack, closely followed by the bias memory component. The feature map trojans resulted in a significant accuracy degradation of 78.16% with a 0.15% and 0.29% increase in Look-Up-Table (LUT) utilization for the SDAT and GDAT variants, respectively. In comparison, the bias trojans caused an accuracy degradation of 63.33% with a LUT utilization increase of 0.20% and 0.33% for the respective trojans. The power consumption overhead was consistent at 0.16% for both the attacks and trojan versions.



## ACKNOWLEDGEMENTS

---

We want to express our gratitude to our supervisors, Dr. Mahdi Fazeli and Dr. Ahmad Patooghy, for their guidance throughout this project.

Kind regards,  
Simon & Marcus  
*Halmstad, Sweden*



# CONTENTS

---

1	INTRODUCTION	1
1.1	Problem Statement	2
1.2	Research Questions	2
1.3	Assumptions & Limitations	2
1.4	Novelty & Contribution	3
2	BACKGROUND	5
2.1	CNNs on FPGAs	5
2.2	Register-Transfer-Level vs Gate-Level	6
2.3	Hardware Trojan attacks	7
2.4	Fixed point representation	7
2.5	FPGA Components	8
3	RELATED WORKS	11
3.1	Attacks	11
3.2	Countermeasures	12
3.3	Summary	15
4	METHODOLOGY AND IMPLEMENTATION	17
4.1	Proposed Method at a Glance	17
4.2	The Proposed Attack	18
4.2.1	Sensitivity Analysis	18
4.2.2	Selecting the Trojan Payload	19
4.2.3	Hardware Trojan Design	19
4.3	Evaluation Method	21
4.3.1	Metrics	21
4.3.2	Comparative Analysis	21
4.4	Model and toolchain	22
4.4.1	LeNet-5 model	22
4.4.2	Quartus	23
4.4.3	MATLAB	23
4.4.4	ModelSim	23
4.5	Memory Structure and Attack Implications	23
4.6	Functional Validation & Retraining	24
4.7	Establishing Baseline Performance Metrics	24
4.8	Implementation	25
4.8.1	Sensitivity analysis	25
4.8.2	Selecting target of trojan	28
4.8.3	Payload of trojan	28
4.8.4	Time to activation of trojan	29
5	RESULTS	31
5.1	Impact of Trojan Attacks on Accuracy	31
5.1.1	Gradually Degrading Accuracy Trojan	31
5.1.2	Suddenly Degrading Accuracy Trojan	31
5.2	Resource Utilization and Overhead	32

5.2.1	Hardware Overhead	32
5.2.2	Power Consumption Overhead	33
5.3	Comparative Analysis with Existing Trojans	34
6	DISCUSSION	37
6.1	Discussion of Results	37
6.2	Feasibility of attack	38
6.3	Future Work	40
6.3.1	Stealthiness to trigger detection methods	40
6.3.2	Attacking other model architectures	41
6.3.3	Intentionally overflowing Trojan	41
6.4	Adressing Research Questions	42
7	CONCLUSION	43
	BIBLIOGRAPHY	45

## LIST OF FIGURES

---

Figure 1	Visualization of the assumed unsafe parts of the design process.	3
Figure 2	Design and implementation phases of a CNN accelerator from CNN model. Adapted from [11].	6
Figure 3	Visualization of original and trojaned circuit.	7
Figure 4	Q1.14 binary fixed-point format.	8
Figure 5	Flowchart of the method.	17
Figure 6	Weight distribution of model.	18
Figure 7	Gradually Degrading Accuracy Trojan.	20
Figure 8	Suddenly Degrading Accuracy Trojan.	20
Figure 9	Visualization of model configuration. From [32].	23
Figure 10	Model accuracy degradation when individual weights are set to 0.9999.	26
Figure 11	Histogram of prediction values of neuron 9.	29
Figure 12	GDATs hardware utilization of total available hardware.	32
Figure 13	SDATs hardware utilization of total available hardware.	33
Figure 14	Histogram of prediction values of neuron 10.	41

## LIST OF TABLES

---

Table 1	Q notation. Adapted from [17].	8
Table 2	Accuracies with different payloads.	19
Table 3	LeNet-5 model configuration.	22
Table 4	Baseline hardware utilization of accelerator.	25
Table 5	The twenty most harmful weights in each layer.	27
Table 6	The ten most harmful biases in each layer	27
Table 7	Payload of trojans.	29
Table 8	Bit-size of counter compared to accuracy impact in Matlab.	30
Table 9	Accuracy of the GDATs compared to baseline.	31
Table 10	Accuracy of the SDAT compared to baseline.	32
Table 11	Hardware utilization of accelerator with GDATs.	33
Table 12	Hardware utilization of accelerator with SDATs.	33
Table 13	Power estimation of accelerator with GDATs.	34
Table 14	Power estimation of accelerator with SDATs.	34
Table 15	Hardware utilization of HTs.	34
Table 16	Power consumption overhead.	35

## ACRONYMS

---

HT	Hardware Trojan
NT	Neural Trojan
IC	Integrated Circuit
NN	Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
FPGA	Field-Programmable Gate Array
RTL	Register-Transfer-Level
HDL	Hardware Description Language
PLD	Programmable Logic Device
PE	Processing Element
MCR	Malicious Category Recognition
TR	Trigger Recognition
ASIC	Application Specific Integrated Circuit
DSP	Digital Signal Processors
LUT	Look-Up-Table
GDAT	Gradually Degrading Accuracy Trojan
SDAT	Suddenly Degrading Accuracy Trojan
TTT	Ticking Timebomb Trojan
ROM	Read-Only Memory
RWM	Read-Write Memory



## INTRODUCTION

---

The increasing applications of neural networks have spurred interest in specialized hardware to enhance performance and efficiency. [FPGAs](#) are gaining popularity in machine learning due to their flexibility, high performance, and efficiency [8, 9, 30]. They allow for the customization of hardware to specific [NN](#) architectures, facilitating optimized computation and faster inference times compared to traditional processors.

However, the high cost of chip design has made it infeasible for companies to manage the entire development cycle in-house. Consequently, the industry has shifted towards a horizontal process model, leading to increased outsourcing of [IC](#) design to third parties [16, 22]. This shift allows for third parties to insert malicious functionalities, such as [HTs](#), during the design phase [2, 16, 18, 22].

If not effectively addressed, [HTs](#) can lead to significant financial losses, unauthorized access to sensitive information, compromise of system integrity through backdoors, performance degradation, or denial-of-service attacks.

Attackers generally use two strategies for triggering a trojan: timer-based and data-based triggers. Timer-based triggers are designed to activate after the machine has been powered on for a certain number of cycles, while data-based triggers are activated by a predefined sequence of data in the input stream. Timer-based triggers offer several benefits, including better deployability, as they do not require post-deployment interaction [27]. Despite this, current research on trojans targeting [CNNs](#) focuses mainly on inducing misclassifications through external triggers [4, 6, 11, 31]. Additionally, there is a lack of investigation on trojans that degrade the overall accuracy of [CNN](#) over time. Gradual accuracy degradation can erode trust in the model's performance, leading users to question its reliability, and potentially resulting in significant economic damages and reputational harm when the degradation is eventually noticed.

Therefore, this thesis will investigate the effects of an accuracy-degrading attack and identify vulnerabilities to such an attack within a [FPGA](#)-based [CNN](#) accelerator. The research will be conducted through the following steps: (1) examining the inherent properties of a [FPGA](#)-based [CNN](#) accelerator at the hardware abstraction level to determine how individual weights, biases, and feature maps can be targeted, (2) maximizing the attack's effectiveness in terms of accuracy degradation while remaining hidden, and (3) conducting a comparative

analysis with other hardware trojans to assess the footprint of the attack.

This thesis is organized as follows: Chapter 1 introduces the problem statement and research questions. Chapter 2 provides the necessary background, detailing the foundational knowledge required to understand the work presented. Chapter 3 reviews relevant literature and identifies the research gap this thesis addresses. Chapter 4 outlines the proposed method, the metrics for evaluation and the implementation of the proposed attack. Chapter 5 presents the results. Chapter 6 discusses the results, the feasibility of the attack, and potential future works. Finally, Chapter 7 presents the conclusions.

### 1.1 PROBLEM STATEMENT

Third-party tools used during the design process could potentially have adversarial intent, and the effects of a trojan being inserted by these tools must be carefully studied. Additionally, while most attacks in this field are input-triggered, the potential impact of an attack that could gradually diminish the functionality of an NN accelerator without requiring an external trigger also warrants further investigation. This investigation should assess the potential impact of such attacks and identify the vulnerabilities where these attacks are most successful and can remain hidden.

### 1.2 RESEARCH QUESTIONS

This study explicitly focuses on the security threat introduced by a novel form of accuracy degradation HTs, specifically targeting hardware accelerators tailored for machine learning applications. The primary objective is to expand the scope of potential attacks against NN accelerators by introducing, implementing, and assessing the viability of such an attack.

To that end, the thesis aims to answer the following research questions:

- From an adversary’s perspective, which components of a neural network accelerator are susceptible to an accuracy-degrading hardware trojan that deteriorates accuracy over time?
- In real-world scenarios, how well would an accuracy-degrading hardware trojan be hidden compared to other types of hardware trojans targeting neural network accelerators?

### 1.3 ASSUMPTIONS & LIMITATIONS

Certain assumptions have been made to narrow the project’s scope. It is assumed that the attack would occur during the synthesis, place

and route, or bitstream generation stages. The tools used in these processes are considered unsafe, implying a potential risk of them inserting a [HT](#) into the design. Figure 1 provides a visual representation of the anticipated attack vector for the project.

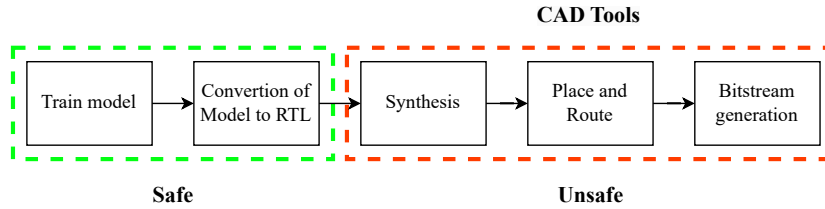


Figure 1: Visualization of the assumed unsafe parts of the design process.

#### 1.4 NOVELTY & CONTRIBUTION

This research project focuses on identifying vulnerabilities in the hardware components of [NN](#) accelerators susceptible to accuracy degrading [HT](#) insertions. Since trojans embedded within [FPGAs](#) can conceal themselves within unused or underutilized resources, the associated hardware and power overhead may differ from that in Application Specific Integrated Circuit ([ASIC](#)) implementations. Although, in theory, the attacks presented in this thesis can be implemented in [ASICs](#), they will not be evaluated in such an environment.

To our knowledge, the type of attack presented in this thesis – an [HT](#) degrading accuracy over time targeting [NN](#) accelerators, has never been done before.

Therefore, the contributions of this paper will be:

- A novel accuracy degrading [HT](#) targeting the hardware concerned with writing and storing of weights, biases, or feature maps within a [NN](#) accelerator.
- An analysis of where [FPGA](#)-based [NN](#) accelerators are vulnerable to [HT](#) insertions by the CAD tools during the synthesis, place and route, or bitstream generation stages.



## BACKGROUND

---

This section aims to provide the essential knowledge to understand the work presented in this thesis. In Section 2.1, the implementation of a CNN on FPGAs is explained, and Section 2.2 will detail the distinctions between Register-Transfer-Level (RTL) and gate-level simulations. Section 2.3 explains the basics of HT attacks, and Section 2.4 introduces the fixed point representation used in this study. Finally, Section 2.5 gives an overview of the core FPGA components.

### 2.1 CNNs ON FPGAS

FPGA platforms are great candidates for accelerating CNNs because of their inherent parallelism. The numerous computational units (convolutions, pooling, etc.) that compose a CNN can be mapped directly onto the FPGA's reconfigurable fabric. This allows many layers to operate simultaneously and parallelizes the computations within each layer. FPGAs achieve this through dedicated hardware blocks for core CNN operations.

Further efficiency gains come from quantization techniques, where numbers are represented with lower precision, reducing memory footprint and speeding up computations within the hardware blocks. Although these blocks are powerful, careful design of the dataflow is critical. A well-designed dataflow efficiently moves image data and intermediate results between blocks to ensure maximum utilization of the FPGA resources.

The design and implementation phases for a typical CNN accelerator is depicted in Figure 2, where a suitable CNN model and training data are selected based on the target application. The model is trained using frameworks such as TensorFlow, PyTorch, etc.

Upon completion of the training phase, the next step involves a thorough analysis of the model and its parameters. This stage, known as the toolchain step, which translates the conceptual architecture of CNN into Hardware Description Language (HDL) code.

Following the translation to HDL, the design is synthesized. The synthesis converts the RTL into a gate-level netlist. Subsequently, the placement and routing step occurs, which involves mapping the logical design onto the array of programmable logic blocks, interconnects, and I/O blocks on the FPGA. Finally, a bitstream file is generated, which can be uploaded onto an FPGA.

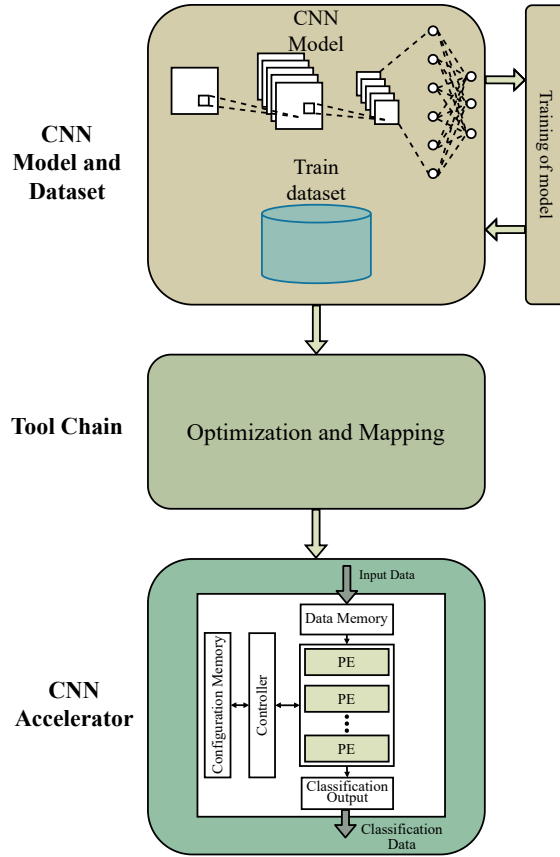


Figure 2: Design and implementation phases of a CNN accelerator from CNN model. Adapted from [11].

## 2.2 REGISTER-TRANSFER-LEVEL VS GATE-LEVEL

RTL and gate-level are two different levels of abstraction used in digital design and simulation. RTL simulation operates at a higher level of abstraction, focusing on the behavior of the digital design at the register transfer level to do functional verification and validation. These simulations are typically faster and require fewer computational resources. Gate-level simulation, on the other hand, operates at a lower level of abstraction, where the digital design is represented using logic gates and flip-flops. These simulations provide a more detailed view of the digital circuit's behavior but are more computationally intensive and slower than RTL simulations. Therefore, the gate-level abstraction level can be used to estimate the power and hardware footprint of the design while the functionality of the design can be simulated on the RTL level.

## 2.3 HARDWARE TROJAN ATTACKS

A **HT** is a malicious alteration or addition to physical hardware or hardware design. The target is the hardware itself or the logic that the hardware executes. This includes **ICs**, processors, **FPGAs**, and the logic or firmware running on them. **HTs** are introduced during the manufacturing process or the design phase, such as by modifying the **HDL** code used to program an **FPGA**. A **HT** consists of a conditional trigger and a payload. The trigger for **HTs** can be varied, including specific signals or time-based triggers. The payload of a **HT** can range from altering the function of the hardware, leaking sensitive information, and degrading system performance, for example, targeted/un-targeted misclassification and accuracy degradation. Figure 3 shows a simple example of a simple circuit with its trojan-inserted counterpart.

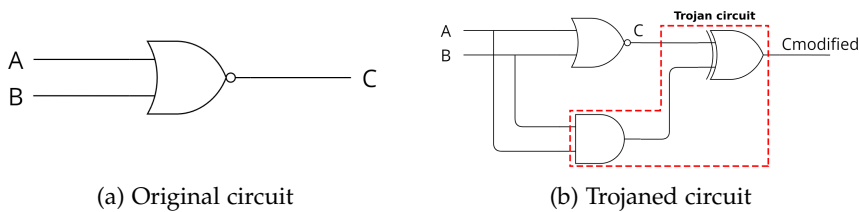


Figure 3: Visualization of original and trojaned circuit.

## 2.4 FIXED POINT REPRESENTATION

The model that will be used in the project is a modified LeNet-5 model written in SystemVerilog. With the model being written in **HDL** there are some limitations within the precision of weights and computations. In Table 1 a list of the different **Q** notations ranging from **Q0.15** to **Q15.0**.

Table 1: Q notation. Adapted from [17].

Format	Largest positive integer	Least negative value	Precision
Q0.15	0.999969482421875	-1	0.000030517578125
<b>Q1.14</b>	<b>1.99993896484375</b>	<b>-2</b>	<b>0.00006103515625</b>
Q2.13	3.9998779296875	-4	0.0001220703125
Q3.12	7.999755859375	-8	0.000244140625
Q4.11	15.99951171875	-16	0.00048828125
Q5.10	31.9990234375	-32	0.0009765625
Q6.9	63.998046875	-64	0.001953125
Q7.8	127.99609375	-128	0.00390625
Q8.7	255.992187	-256	0.0078125
Q9.6	511.984375	-516	0.015625
Q10.5	1023.96875	-1024	0.03125
Q11.4	2047.9375	-2048	0.0625
Q12.3	4095.875	-2096	0.125
Q13.2	8191.75	-8192	0.25
Q14.1	16383.5	-16384	0.5
Q15.0	32767	-32768	1

The weights of the model are represented in the Q1.14 binary fixed-point format as seen in Figure 4. In this format, two bits are dedicated to the integer part, where one is the signed bit. The remaining 14 bits of the 16-bit field are dedicated to the fraction part [21].

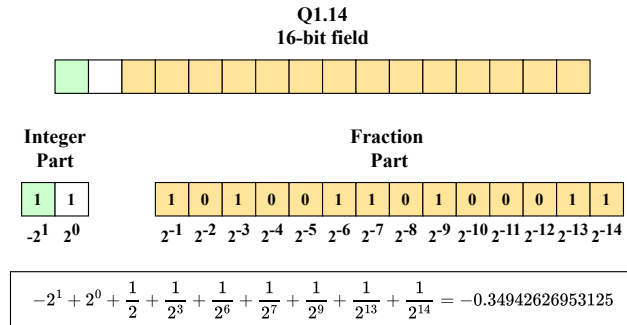


Figure 4: Q1.14 binary fixed-point format.

## 2.5 FPGA COMPONENTS

At the core of an FPGA's architecture are the logic cells, which are the basic building blocks for digital designs. These logic cells handle both combinational and sequential logic functions. Each logic cell includes a LUT and a register, among other parts. Logic cells are connected through a network of programmable routing resources, allowing for the creation of complex digital circuits. This combination

of programmable interconnects and configurable logic cells gives the [FPGA](#) its reconfigurability and versatility.

[LUTs](#) are an essential part of the logic cells. They function as small truth tables programmed to implement any desired boolean function, like NAND, XOR, etc. During the [FPGA](#) configuration process, these [LUTs](#) are programmed to perform the specific logic functions needed for the application, providing the core functionality of the combinational logic.

Registers are used within logic cells to handle sequential logic. They temporarily store data and are crucial for creating state machines, pipelining data, and other sequential processes, such as storing and updating the state of variables that change over time.

Overall, the combination of [LUTs](#) for combinational logic, registers for sequential logic, and programmable interconnects for flexible routing forms the backbone of [FPGA](#) architectures. This setup enables the [FPGA](#) to adapt to various applications, from simple logic functions to complex digital systems, making it an important platform in modern electronic design.



## RELATED WORKS

---

This chapter explores the existing body of work related to security in **NNs**, with a specific focus on the hardware perspective. The section is structured as follows: In Section 3.1, various attacks will be explored, including Neural Trojan (**NT**) and **HT** attacks. Section 3.2 will address the current countermeasures against these attacks. Finally, Section 3.3 presents a summary of the related works.

### 3.1 ATTACKS

A threat faced by **NNs** is **NTs**. In **NT** insertion attacks, the attacker's aim is usually not to degrade the model's performance but rather to hide malicious functionality that can be triggered by specific patterns present in the input to the model. Not degrading the model's overall performance is actually very important, as it hides the fact that the model has been infected. The hidden functionality implanted could be misclassifying specific inputs as a different target class. According to the Liu et al. [20], the most commonly researched way to insert the trojan functionality is by training data poisoning.

Training data poisoning involves adding a few malicious training samples, including triggers, into the regular training dataset. That way, the model is taught to associate data with the trigger in it with the chosen target class. Since the model is also trained on the majority of clean training data, data without the trigger would still be classified correctly.

Instead of using training data poisoning that hides the **NT** functionality in the weights, **NT** functionality could also be realized by attacking the computing operations used by layers in a model. Clements and Lao [5] propose targeting the computing operations of the neurons to inject malicious functionality into a trained **NN**. Their proposed method involves a stepwise approach: choosing a specific layer as the target and then iteratively calculating how the target operation within that layer should change to generate the malicious output. In essence, this approach can be used to identify the operation modification that produces the perturbation required for either a targeted or untargeted version of a misclassification attack. The same authors investigate the practicality of their proposed method in a later paper [4], where they realize the attack using **HTs** to modify computing operations of different layers to induce the required perturbation for both the targeted and untargeted versions of the attack. **HTs** are malicious alterations to the circuitry, which consists of a trigger and a payload.

The trigger is activated under rare conditions and activates the payload, achieving malicious functionality. In a similar approach, they implement an HT into the activation function of one layer. The trigger and payload are designed using simple logic gates and implemented into the activation function block [6].

Ye et al. [31] implant a HT within a specific Multiply and Adder-Tree in the first convolution layer of an FPGA-based CNN accelerator. This setup enables adversaries to manipulate the CNN's classification output by making minor input image adjustments, activating the HT. The trojan leverages two key modules: the Malicious Category Recognition (MCR) and the Trigger Recognition (TR) modules. The TR module is responsible for assessing the operational status of the accelerator. If the HT is not activated, the accelerator functions normally, processing input images without interference. However, when the HT is triggered, it switches the output category to a malicious one predefined by the MCR module.

Instead of focusing on computing operations within Processing Elements (PEs) like Ye et al., Hou et al. [11] created a method for launching an attack to take advantage of a HT within the reconfigurable interconnection network of an FPGA-based Deep Neural Network (DNN) accelerator. This attack aims to alter the data pathway upon activation of the HT. As a result of this alteration, the interconnection of the PEs potentially produces erroneous computations.

A HT, as presented by Li and Hou [19], deviates from typical trojans, which are usually triggered by input images. In the DNN accelerator, the processing elements are reused across layers and upon completion of one layer, an interrupt signal is sent to the host CPU. This signal serves as an indication that intermediate results from the next layer are ready for processing. The trojan leverages this signal since different model configurations will lead to differing sequences of interrupt signals. Thus, model parameters can be used as a triggering condition. Once this triggering condition is met, the trojan is activated.

The payload of this trojan specifically targets the global bias buffer and modifies the values of the bias register to prevent the activation of intermediate data. Consequently, the neurons become inactive and the neural network forward-propagation stops. This disruption leads to the DNN accelerator malfunctioning.

### 3.2 COUNTERMEASURES

In the context of ICs, defending against HTs is a critical concern to ensure the security of electronic systems. In a comprehensive survey [7], Dong et al. explores strategies to detect and thwart HTs insertions. It is important to note that the protective techniques discussed herein are primarily tailored for general IC security and are not explic-

itly targeted at hardware accelerators for NNs. However, the insights obtained from these approaches can offer valuable perspectives on improving the security against HTs on hardware accelerators for NNs.

Dong et al. categorizes the HT protection scenario into two main domains: HT detection and HT prevention.

From the detection domain, with a focus on pre-silicon design, the authors present auxiliary detection, runtime monitoring, and static detection. Auxiliary detection, for example, improves the effectiveness of logic testing and channel analysis by comparing HT-free ICs and HT-infected ICs. Meanwhile, runtime monitoring operates continuously during IC usage, making it capable of detecting HTs throughout the IC's operational lifecycle, although at the cost of extra performance overhead. Moreover, the static detection strategy examines specific features in-circuit parameter information, employing these features to train a machine learning model that distinguishes between clean and infected ICs.

As a proactive measure to prevent HT insertions, the authors introduce *design for trust*. This approach incorporates security measures from the onset of the IC design process. In terms of HT defense, logic locking and 'obfuscation and filling' are utilized to conceal the function of the IC or fill out unused space in the design to minimize the attack area, thereby complicating attackers' efforts to inject HTs.

Instead of developing a universal protective measure, Trippel et al. [25] recommend adopting a divide-and-conquer approach to detecting HTs. This strategy involves breaking down the trojan design space and methodically eliminating each type of trojan. They define a category of hardware trojans termed Ticking Timebomb Trojans (TTTs), characterized by triggers that gradually approach activation as the system continues to operate. These triggers are built around a non-repeating sequence counter that increments following specific events within the hardware. Moreover, they developed Bomberman, a dynamic trojan verification framework designed to identify potential TTTs within a design's HDL. This is achieved by monitoring the sequences handled by all state-saving components – the critical elements in TTTs that manage and monitor the time counter's triggering state. In practical tests, Bomberman successfully detected all TTT variants across four real-world hardware designs, significantly outperforming previous methods with a false positive rate of less than 1.21 %.

Current countermeasures against neural trojans in NNs are summarized by Kaviani and Sohn in their survey [15]. They empathize that the current focus within the research community is primarily on detection and mitigation techniques. Detection occurs when the defenders try to identify suspicious inputs using anomaly detection techniques or try to detect the presence of trojan in the NNs by searching for specific indicators of trojan attacks. Detection of NTs is there-

fore divided into subcategories in the survey paper [15], and some of those subcategories will be introduced below.

Malicious input detection occurs when defenders try to identify suspicious inputs using anomaly detection techniques. One such way is introduced in a paper [3] by [Chen et al.](#). The intuition behind their proposed method is that the reason why a clean input gets its classification will differ from an input that contains the trigger and gets the same classification. This can be determined by looking at the activations of the last hidden neuron layer when querying the model with training data. Suppose that a particular label in the training data has been poisoned. In that case, the resulting activations for that label observed during the queries will result in two distinct clusters. One is the accurate classification caused by the class features and the other is the classification caused by the presence of the trigger.

Another subcategory of [NT](#) detection is trigger detection. The core concept of trigger detection comes down to the fact that [NT](#) insertion through training data poisoning will create a shortcut to the feature space of the target label. In other words, a large input perturbation is not required to change the original label to the target label. This attack feature can be used to defend against it, which is what [Wang et al.](#) does in their paper [28]. Their approach focuses on determining how much modification (perturbation) is required for the model to misclassify any input as another label. When this has been done for all classes as the target label, outlier detection can be used to determine if a specific label requires less input modification to be misclassified as. If an outlier is found, the trigger has basically been reverse-engineered since the defender knows the required perturbation. The authors use this fact to propose mitigating the [NTs](#) by first identifying the infected neurons active when the trigger is present. Then, they could either filter out inputs that activate those neurons or prune the infected neurons.

Model-level detection involves the classification of an entire model as either trojaned or clean without relying on specific input data. One method of this is presented by [Xu et al.](#) in their paper [29]. The authors introduce a technique where a classifier is trained to determine if a given target model contains a trojan based on some properties of the model itself. The first step involves training shadow models. These are clean or trojaned models trained on the same task as the target model. The classifier can then be trained on the clean and trojaned shadow models and, lastly, be used to predict which class a target model belongs to.

The mitigation methodologies highlighted in the paper [15] can also be categorized further into specific techniques.

Neuron Pruning involves the removal of neurons from the [NN](#). This process can be carried out arbitrarily without specific knowledge of

the trojaned neurons. Alternatively, it may only remove neurons identified as compromised, as demonstrated by Wang et al. [28].

Unlearning entails retraining the model to achieve accurate classification even when exposed to triggers.

Filtering involves passing inputs through a filter mechanism designed to exclude inputs containing detected triggers. On the other hand, pre-processing functions similarly by eliminating triggers from inputs and preventing the trojan from activating.

While existing research delves into preventive techniques against HTs in general ICs, the scenario changes when considering specialized methods to counter HTs in NN accelerators. The only recent research into this specific area was by Hou et al., who in their work [11] developed a detection technique to protect the reconfigurable interconnection network from HTs. The countermeasure, based on physical unclonable functions, serves a dual purpose: it can detect the trojan and identify their specific locations within the system.

### 3.3 SUMMARY

Recent literature reveals that NNs, including their hardware designs for FPGAs, are vulnerable to various forms of attacks, such as NTs and HTs. While numerous methods have been suggested to counter HT threats, a complete solution capable of identifying and neutralizing all potential trojans remains elusive. Specifically, the field lacks robust defensive measures against trojans aimed at NN accelerators.

Current attacks are triggered primarily on the basis of data, such as those detailed in [4, 6, 11, 31]. The works [4, 31] focus on computational operations within convolutional layers, while [6] target activation function blocks, and [11] manipulate the data paths of PEs used in convolutional computations, leading to incorrect outcomes. Although these attacks are well-concealed within the designs, they typically impact only a single prediction when triggered. In contrast, a timer-based attack discussed in [19], when triggered, completely disables the pre-set model of the accelerator from producing any predictions.

There is still a gap in research regarding attacks that gradually reduce accuracy over time. While previous studies have focused on computational operations or reconfigurable networks, this study suggests a novel approach. It explores an attack that decreases accuracy over time by targeting hardware responsible for storing and manipulating weights, biases, and feature maps. Furthermore, the results of this attack can offer insight into the specific vulnerabilities of neural network accelerators against such attacks, considering the overhead introduced and whether the trojans remain undetected during functional testing.



## METHODOLOGY AND IMPLEMENTATION

---

This chapter describes the methodology used in conducting this thesis project. First, Section 4.1 introduces the proposed method. Section 4.2 explains the weight sensitivity analysis and proposed attack strategy. Following that, Section 4.3 details the evaluation methods used in the project. Next, Section 4.4 describes the model and tools used. Section 4.5 explains the memory structure of the weights and biases in the attacked network. Section 4.6 presents the functional validation and retraining process. Section 4.7 discusses the establishment of baseline metrics. Finally, Section 4.8 covers the implementation.

### 4.1 PROPOSED METHOD AT A GLANCE

Addressing the first research question outlined in Section 1.2, this thesis conducts a sensitivity analysis of the CNN model to identify the most impactful weights and biases. Based on this analysis, a trojan is implemented to degrade the accuracy of the accelerator over time. The trojans implemented will be evaluated using the metrics defined in Section 4.3. These metrics will determine the most susceptible components among those observed.

To respond to the second research question regarding how effectively our trojan remains concealed compared to other trojans, we will evaluate the hardware and power consumption overheads introduced by our trojan and compare these to those associated with other state-of-the-art hardware trojans. Figure 5 shows a flow diagram of the proposed method.

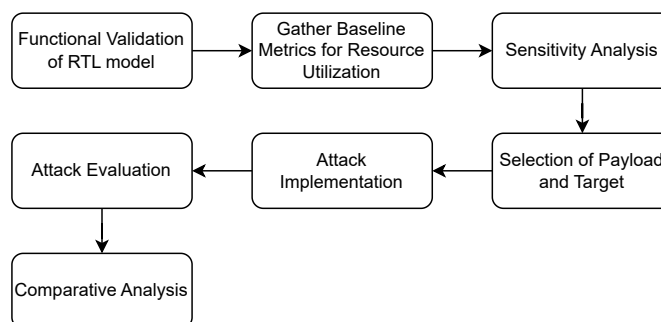


Figure 5: Flowchart of the method.

## 4.2 THE PROPOSED ATTACK

The proposed attack is a white box attack that requires knowledge of model architecture, weights, and biases. Specifically tailored for CNNs running on embedded platforms, it capitalizes on the unique characteristics of neural networks, such as weights and biases, while navigating the limitations imposed by the fixed Q1.14 resolution used in embedded systems. What makes this attack particularly dynamic is its adaptability. By leveraging a sensitivity analysis, multiple potential target weights or biases can be identified. Moreover, the behavior of the HT can be tailored to suit the attacker’s specific objectives. The outcome of the attack can also be customized, attackers aren’t restricted to targeting weights or biases that cause the most significant drop in accuracy. For example, they can opt for a payload where only a 10% reduction in accuracy occurs when the attack is fully executed.

The attack can be divided into three stages: [Sensitivity Analysis](#), [Selecting the Trojan Payload](#), and [Hardware Trojan Design](#). These three stages are described below.

### 4.2.1 Sensitivity Analysis

The first part of the attack aims to identify the best weights or biases to target. This is done by conducting a sensitivity analysis, where one weight at a time is set to 0.9999, and the resulting accuracy over 100 images is recorded. While simulating only 100 images is not sufficient to determine an exact accuracy, it is enough to gauge whether a specific weight or bias is important.

Weights are set to 0.9999 because this value is significantly higher than the weight distribution, shown in Figure 6, making the impact of high-importance weights on accuracy more noticeable. At the same time, it is small enough to avoid overflowing the Q1.14 format.

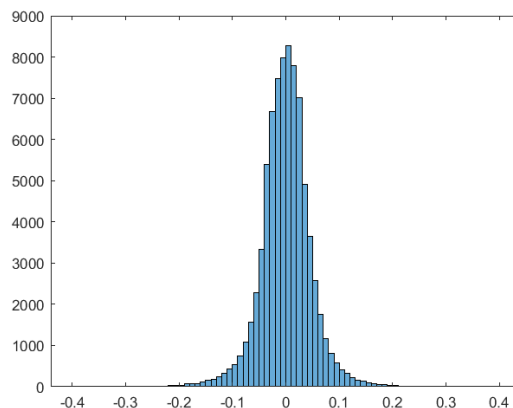


Figure 6: Weight distribution of model.

### 4.2.2 Selecting the Trojan Payload

One of the defining characteristics of this attack is the customizable outcome of the attack. While the result from the sensitivity analysis is an excellent indicator of potential targets, it doesn't show the range of possibilities each of those targets offers. For example, Table 2 shows the effect of an attack on bias 9 ( $B_9$ ) in the last fully connected layer with different payloads.

Table 2: Accuracies with different payloads.

Bias 9 payload	Resulting accuracy
None	99.00%
0.1	99.80%
0.2	98.20%
0.3	97.50%
0.4	95.80%
0.5	91.80%
0.6	84.40%
0.7	73.90%
0.8	61.20%
0.9	51.10%

Therefore, the second part of the attack is to decide on the desired outcome and select a payload that achieves that while avoiding overflowing the Q1.14 format.

### 4.2.3 Hardware Trojan Design

Through the sensitivity analysis, weights and biases that are particularly sensitive are pinpointed. This sensitivity also implies that the resultant feature maps from these weights and biases operating on an input are vulnerable and can be targeted. Therefore, this study will examine attacking three different components of the CNN accelerator: the components responsible for storing and writing the weights, biases, and feature maps. To address the dynamic nature of the attack and ensure broad application coverage, two versions of accuracy degradation trojans, sudden and gradual, will be implemented.

#### 4.2.3.1 Gradually Degrading Accuracy Trojan

The Gradually Degrading Accuracy Trojan, as its name indicates, achieves its effect by gradually increasing an offset added to the target weight, bias, or feature map. This offset is incremented by the minimum offset that the Q1.14 format allows, ensuring a slow but steady degradation of accuracy until the desired payload is reached. At the same time, it is crucial for the target to be incremented slowly enough so that

no significant accuracy drop is noticed during functional testing. In Figure 7, a **GDAT** is depicted, showing a slow and gradual decrease in accuracy compared to the non-trojan **CNN**.

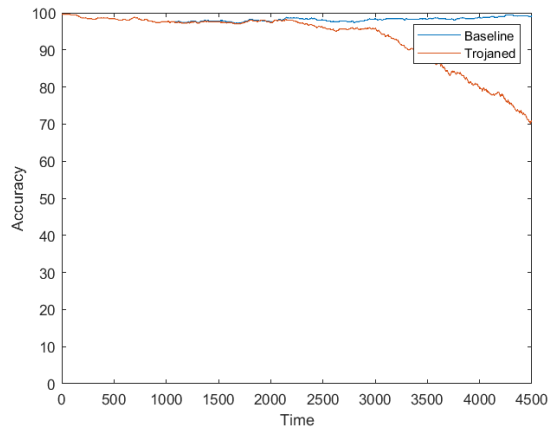


Figure 7: Gradually Degrading Accuracy Trojan.

#### 4.2.3.2 Suddenly Degrading Accuracy Trojan

The Suddenly Degrading Accuracy Trojan, compared to the **GDAT**, will lay dormant for a long time and have no impact at all during functional testing. When it activates, a large offset is added to the target weight, bias, or feature map, causing a sudden drop in accuracy. In Figure 8 below, a **SDAT** trojan is depicted, showing an abrupt drop in accuracy compared to the non-trojan **CNN**.

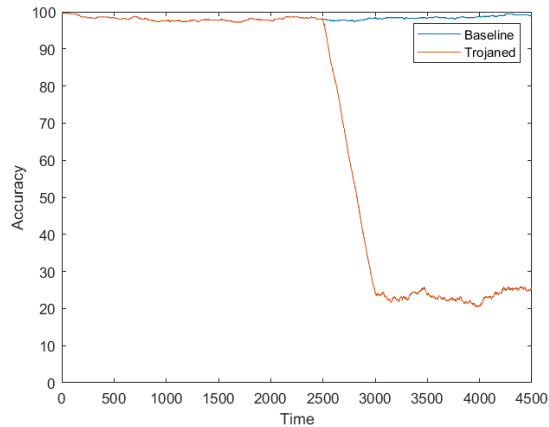


Figure 8: Suddenly Degrading Accuracy Trojan.

## 4.3 EVALUATION METHOD

This section outlines the specific metrics used to evaluate the effectiveness of the proposed trojan attacks and presents a comparative analysis with other state-of-the-art HT attack methods against NN accelerators.

### 4.3.1 Metrics

Three critical factors must be considered to assess the effectiveness of the accuracy-degradation HTs: stealthiness, resource overhead, and accuracy impact. Initially, it is crucial to determine whether the compromised accelerator can pass the functional testing phase, which verifies that the hardware meets functional specifications. Secondly, the overhead introduced by the trojan should be minimal to evade detection and maintain the functionality and performance of the accelerator. Finally, the long-term effectiveness of the trojan is measured by the extent of accuracy degradation observed when the accelerator operates in real-world applications.

By analyzing these metrics in combination, the most vulnerable component among those targeted can be identified. The evaluation will focus on the following metrics:

- **Stealthiness**

To assess the attack's stealthiness, the trojan will be embedded in the design and undergo a two-week functional testing phase. In this simulation environment, the CNN accelerator is limited to 30 samples each second, which results in 36,288,000 samples during the functional testing phase.

- **Resource Overhead**

Examining the differences in power consumption and hardware utilization between the original and trojan-inserted RTL models. The objective is to minimize the increase in both hardware and power consumption.

- **Accuracy Impact**

The accuracy of the trojan-inserted model is compared to the original RTL model over an extended period to evaluate the long-term impact of the attack.

### 4.3.2 Comparative Analysis

Given that this attack introduces a novel behavior with different objectives and outcomes compared to existing HTs targeting NN accelerators, direct comparisons in terms of stealthiness and accuracy impact

are challenging. Specifically, the attack features a "once triggered, always on" time-based trigger that degrades the accuracy of the NN accelerator. Unlike other trojans that have data-based triggers and aim to misclassify samples only when activated, a decline in accuracy is caused by the proposed trojans, either suddenly (SDAT) or gradually over an extended period (GDAT).

Therefore, the comparative analysis is limited to the resource overhead introduced by the trojans, which provides a common ground for evaluation across different types of attacks. This focus allows the additional resources required by the proposed trojans to be objectively assessed in comparison to others, despite variations in their attack behaviors and goals.

#### 4.4 MODEL AND TOOLCHAIN

This section provides a brief overview of the tools utilized for the implementation and experiments conducted during this project, including the model itself, as well as the tools employed for simulation and synthesis.

##### 4.4.1 LeNet-5 model

The experiments will use a slightly modified 7-layer LeNet-5 model found online [32]. This model uses 16-bit fixed-point resolution in the Q1.14 format and is written in SystemVerilog. The model is trained on the MNIST dataset for handwritten digit classification. Table 3 shows the model architecture and parameters.

Table 3: LeNet-5 model configuration.

Layer	Dimension	Filter Size	Feature Maps
Input	30x30	-	-
Conv1	28x28	3x3	16
AvgPool1	14x14	2x2	-
Conv2	12x12	3x3	32
AvgPool2	6x6	2x2	-
FC1	64x1	-	64
FC2	10x1	-	10

Figure 9 visually represents the model from input to output.

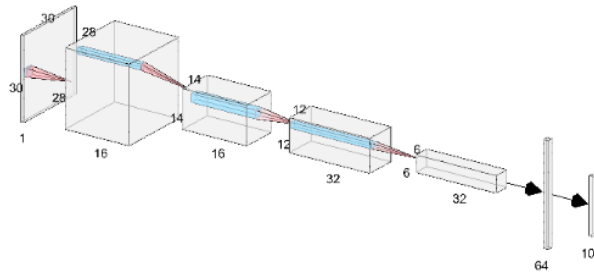


Figure 9: Visualization of model configuration. From [32].

#### 4.4.2 *Quartus*

Quartus is a versatile Programmable Logic Device (PLD) design software enabling designers to analyze, synthesize, and compile their HDL designs [14]. It facilitates various stages of the design and testing process, including simulations and programming onto target devices. Quartus additionally provided data for the power and hardware overhead evaluation metrics.

#### 4.4.3 *MATLAB*

MATLAB is a high-level programming language created by MathWorks, which is widely used for numerical computing, visualization, and programming. It is particularly good at matrix manipulations and plotting data. For this project, a CNN library available in MATLAB has been used to perform faster high-level simulations of the CNN [26].

#### 4.4.4 *ModelSim*

ModelSim is a HDL simulation and verification tool used in digital design and electronic engineering. It enables designers to simulate and validate Verilog, VHDL, and SystemVerilog designs before hardware implementation. It supports both RTL and gate-level simulations [24], which is essential to be able to calculate the desired accuracy impact evaluation metric.

### 4.5 MEMORY STRUCTURE AND ATTACK IMPLICATIONS

Weights and biases are stored in separate Read-Only Memory (ROM) blocks. Since there are significantly more weights than biases, the implementation divides the weights into 16 different memory blocks, with an additional block for biases. Specifically, 8 of the weight memory blocks store all weights related to the two convolution layers and the last fully connected layer, while the remaining 8 blocks store all

weights for the first fully connected layer. In contrast, feature maps are stored in Read-Write Memory (*RWM*) because new feature maps are generated each time a new image is processed and these memory blocks are thus constantly being written and read from. Based on this storage scheme, an attack targeting a weight, a bias, and a feature map will be implemented to target the three different components associated with the storage and writing of weights, biases, and feature maps.

While it is likely that other implementations use *ROM* for weight and bias storage and *RWM* for feature map storage, this cannot be guaranteed. Additionally, other implementations may structure their memory differently, such as using fewer but larger memory blocks or consolidating each layer's weights and biases into a single block. Furthermore, another implementation might use a different *Q* format for binary number representation. Therefore, while the attacks presented in this thesis could potentially be replicated on another *NN* implementation with some adjustments, they are not directly transferable to another accelerator implementation.

#### 4.6 FUNCTIONAL VALIDATION & RETRAINING

An initial assessment was conducted in a non-adversarial environment. The goal of this evaluation was to verify that the *HDL* model implementation was working correctly. This involved extracting the weights from the *HDL* implementation and converting them from *Q1.14* fixed-point notation to 32-bit floating-point representation. These converted weights were then tested within a LeNet-5 architecture using Matlab to ensure functional equivalence. The tests in Matlab yielded an accuracy of 73.7%, matching the accuracy seen in *RTL* simulations and thus validating the *HDL* model implementation.

With this validation, the model was subsequently retrained using TensorFlow [1] on the MNIST training set to enhance model performance, resulting in a significant accuracy increase to 98.91%. The refined weights were then converted back into *Q1.14* notation and integrated into the original *HDL* code. Verification of the updated model was performed by running *RTL* simulations in ModelSim, using the validation samples from the MNIST dataset. The performance metrics obtained corresponded closely to those observed during the Matlab simulations with an accuracy of 98.697%.

#### 4.7 ESTABLISHING BASELINE PERFORMANCE METRICS

The *RTL* design was synthesized using Quartus Prime v20.1 on a Cyclone IV EP4CE115 *FPGA* [12] to establish baseline performance metrics and resource utilization of the neural network accelerator.

Table 4 shows the hardware usage of the accelerator without any trojan present. This will be used as the baseline to measure the additional hardware overhead introduced by the proposed attack.

Table 4: Baseline hardware utilization of accelerator.

Cyclone® IV EP4CE115	LUTs	Registers	Memory bits	DSP elements
Available	114 480	114 480	3 981 312	532
Baseline	36 260	37 071	2 192 896	288
Usage	31.67 %	32.38 %	55.08 %	54.14 %

The baseline power consumption of the accelerator when no trojan is present is 1.237 W, as estimated by the Intel Early Power Estimator for a Cyclone IV running at 125 MHz [13].

## 4.8 IMPLEMENTATION

This section will introduce all the steps leading up to the insertion and evaluation of the trojans.

### 4.8.1 Sensitivity analysis

Figure 10 presents the accuracy loss of weights in each layer of the neural network when individually set to 0.9999. Each pie chart corresponds to a separate layer within the network, with segments representing the proportion of weights that contributed to various levels of accuracy degradation. Initially, the accuracy of the 100 samples was 100 %. This visualization highlights the sensitivity of the model to large perturbations in weights across different layers, thereby identifying critical weights.

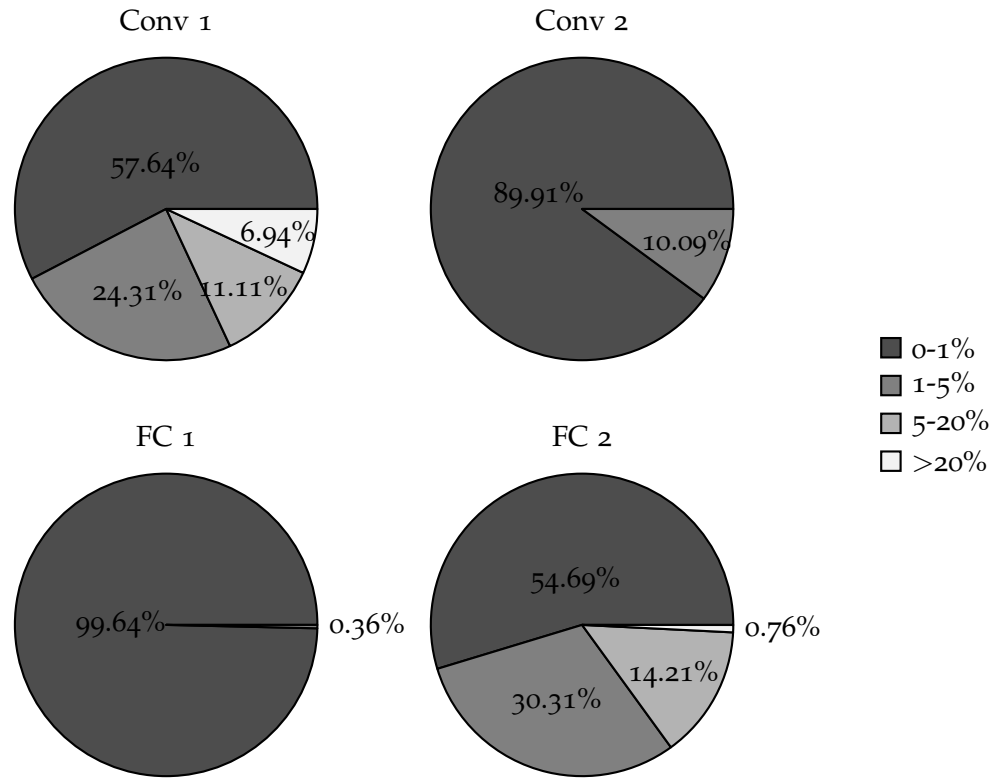


Figure 10: Model accuracy degradation when individual weights are set to 0.9999.

In Table 5, the weights with the most harmful effects on accuracy in each layer are illustrated. In the first convolutional layer, the weights in kernels  $K_{4,5,15}$  significantly impact accuracy, reducing it to as low as 41%. In contrast, the weights in the second convolutional layer seem to have a lesser effect, with the lowest recorded accuracy being 96%. In the second fully connected layer,  $W_{49}$  in neuron 9 ( $N_9$ ) substantially decreases accuracy to 56%, while other weights in neurons  $N_{3,4,9,10}$  also affect accuracy, although to a lesser extent.

Table 5: The twenty most harmful weights in each layer.

Conv1			Conv2			FC1			FC2		
Kernel	Weight	Accuracy	Kernel	Weight	Accuracy	Kernel	Weight	Accuracy	Neuron	Weight	Accuracy
15	9	0.41	210	1	0.96	41	486	0.97	9	48	0.56
15	8	0.5	210	3	0.97	8	9	0.98	10	48	0.73
15	6	0.56	210	5	0.97	8	304	0.98	3	48	0.78
15	7	0.61	1	2	0.98	9	3	0.98	9	9	0.79
15	2	0.69	1	3	0.98	9	4	0.98	9	36	0.79
15	5	0.75	1	4	0.98	9	11	0.98	6	9	0.8
4	1	0.76	1	5	0.98	9	14	0.98	4	48	0.81
4	2	0.76	1	6	0.98	9	17	0.98	8	9	0.81
15	4	0.76	1	9	0.98	9	18	0.98	3	9	0.82
15	3	0.77	2	2	0.98	9	82	0.98	10	9	0.83
4	5	0.8	2	3	0.98	9	83	0.98	10	33	0.83
4	4	0.82	2	4	0.98	9	86	0.98	10	36	0.85
4	6	0.83	2	5	0.98	9	88	0.98	10	55	0.85
15	1	0.83	2	6	0.98	9	89	0.98	6	48	0.86
4	3	0.86	2	8	0.98	9	90	0.98	10	39	0.86
4	9	0.86	2	9	0.98	9	95	0.98	10	63	0.86
5	3	0.86	3	2	0.98	9	96	0.98	1	48	0.87
4	7	0.89	3	3	0.98	9	117	0.98	9	10	0.87
4	8	0.89	5	1	0.98	9	121	0.98	9	55	0.87
5	6	0.89	5	2	0.98	9	123	0.98	10	21	0.87

In Table 6, the biases with the most harmful effects on accuracy in each layer are illustrated. While the bias impacts the accuracy even more than a single weight could, each bias belongs to a kernel. Therefore, it makes sense that the kernels or neurons to which the most dangerous weights belong are correlated to the most dangerous biases. As mentioned earlier, the top twenty most dangerous weights for the first convolutional layer all came from kernels  $K_{4,5,15}$ , which also are the most dangerous biases for the first convolutional layer.

Table 6: The ten most harmful biases in each layer

Conv1		Conv2		FC1		FC2	
Bias	Accuracy	Bias	Accuracy	Bias	Accuracy	Bias	Accuracy
4	0.16	1	0.54	59	0.96	9	0.45
15	0.17	14	0.55	8	0.97	10	0.48
5	0.19	25	0.69	13	0.97	3	0.62
10	0.20	3	0.76	41	0.97	4	0.63
8	0.23	26	0.77	49	0.97	8	0.63
3	0.47	18	0.82	63	0.97	5	0.65
16	0.70	8	0.84	2	0.98	1	0.71
7	0.92	6	0.87	9	0.98	6	0.72
9	0.96	17	0.87	12	0.98	7	0.73
11	0.97	4	0.88	14	0.98	2	0.75

The Figures and Tables above clearly illustrate that the percentage of weights significantly impacting accuracy is remarkably low, particularly in the second convolutional layer and the first fully connected layer.

### 4.8.2 Selecting target of trojan

After conducting the sensitivity analysis, the most harmful weights and biases in each layer are identified. To observe the maximum impact of the trojans, parameters causing the most substantial accuracy reduction are selected for further analysis. Specifically,  $W_9$  in  $K_{15}$  of the first convolutional layer, lowering accuracy to 41 %, and  $B_9$  in the second fully connected layer, reducing accuracy to 45 %, are targeted. For the feature map attack, the 15th feature map ( $Fm_{15}$ ) of the first average pool layer is chosen based on the sensitivity of the weights and bias used to create that feature map.

### 4.8.3 Payload of trojan

In further analyzing the payloads of the selected targets, it is important to note that the accuracy obtained from the sensitivity analysis serves as a baseline and does not represent the theoretical maximum degradation achievable for a specific target. The trojans implemented in this thesis will attempt to reduce the accuracy as much as possible, and thus, the maximal payload that does not overflow calculations will need to be estimated.

For the first target, a weight in the first convolutional layer, the theoretical worst-case scenario is when all positive weights in a filter are multiplied by 1, and all negative weights are multiplied by 0. The target weight is located in  $K_{15}$ , which looks like this:

$$K_{15} = \begin{vmatrix} 0.1694 & 0.0687 & 0.2518 \\ 0.0084 & 0.1072 & -0.0043 \\ -0.2630 & -0.2125 & -0.2501 \end{vmatrix}$$

One can determine the maximum increase of a specific weight such that the worst-case result does not exceed 1.9999, by setting the weight of interest to 0 and using the following equation.

$$W_{Max} = 1.9999 - \sum W_+ - B \quad (1)$$

Since the chosen target is weight  $W_9$  and the bias for  $K_{15}$  is 0.0321, the maximum payload for weight  $W_9$  is:

$$W_{9Max} = 1.9999 - 0.6055 - 0.0321 = 1.3623 \quad (2)$$

For the bias attack, the amplitude of the prediction values for the corresponding neuron is investigated. Figure 11 is a histogram of prediction values for neuron  $N_9$ , where the red corresponds to the values when number 8 ( $N_9$ ) was predicted and blue corresponds to when number 8 was not the prediction.

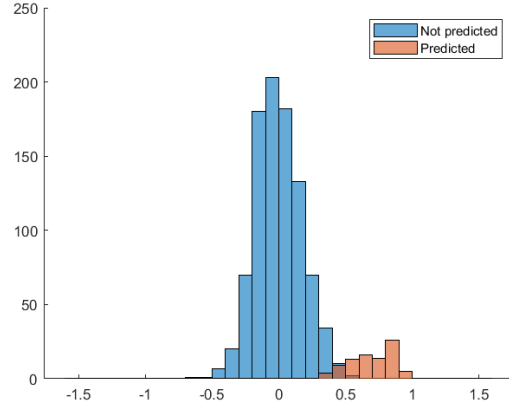


Figure 11: Histogram of prediction values of neuron 9.

From these values, the maximum payload for  $B_9$  without overflowing the Q1.14 format was calculated to be 1.045.

Lastly, the maximum payload for the feature map attack was decided by conducting simulations to figure out the most significant value that did not overflow calculations in the following layers. That payload was estimated to be 0.45. With higher values, a significant difference in accuracy between MATLAB and RTL simulations was seen, indicating that overflows occurred.

In summary, Table 7 shows the payloads chosen for the trojans demonstrated in this thesis.

Table 7: Payload of trojans.

Target	Layer	Payload
Weight 9, kernel 15	Conv 1	1.3623
Bias 9	Fc 2	1.045
Fmap 15	AvgPool 1	0.45

#### 4.8.4 Time to activation of trojan

In the assumed scenario, the functional testing duration is set at two weeks. During this period, the accuracy should remain closely consistent with the baseline accuracy. However, after this period, the accuracy may begin to degrade more noticeably.

Given the accelerator's assumed rate of predicting 30 samples per second, the trojan must process approximately 36,288,000 samples before significantly impacting the accuracy to pass the functional testing phase.

#### 4.8.4.1 Gradually Decreasing Accuracy

Due to the limitations imposed by the Q1.14 format, the **GDAT** increments the payload by one bit, equivalent to  $2^{-14}$ . The hardware overhead for this trojan largely depends on the size of the counter, which increases the payload each time it reaches its maximum value. To increase the chance of the trojan remaining undetected, this counter should be as small as possible, and the accuracy is not allowed to deviate more than 1% from the baseline 98.91% during the functional testing phase. Considering the impracticality of conducting 14 days of continuous simulation, the trojan is instead simulated over 10,000 validation samples from the MNIST dataset. The resulting counter size required to not go above 1% accuracy loss is then scaled by multiplying the counter size in each test by 3628.8, reflecting the ratio of the actual functional testing duration to the simulation period.

Table 8 below showcases the minimum bit-size required for a counter such that attacks pass functional testing. This is the minimum value of the counter, meaning it is also the fastest-acting version of the trojan. Opting for a larger counter size could make passing functional testing easier, albeit at the expense of increased hardware usage.

Table 8: Bit-size of counter compared to accuracy impact in Matlab.

Layer & Parameter	C <sub>1</sub> , K <sub>15</sub> , W <sub>9</sub>	AVG <sub>1</sub> , FM <sub>15</sub>	FC <sub>2</sub> , B <sub>9</sub>
Counter bit-size	Accuracy after functional testing		
15	98.87%	98.46%	98.89%
14	98.86%	95.98%	98.85%
13	98.72%	79.62%	98.73%
12	96.96%	52.02%	97.12%

#### 4.8.4.2 Suddenly Decreasing Accuracy

The time to activation for the **SDAT** was decided to be six months. Based on the assumption of samples per second mentioned earlier, this translates to the **CNN** needing to process 473,040,000 samples before the trojan activates. Consequently, the trojan necessitates a 29-bit counter to accommodate this duration. In contrast, if the trojan were to activate after two weeks, a 26-bit counter would suffice.

## RESULTS

---

This chapter presents the results obtained throughout the course of the project. First, Section 5.1 discusses the attack effectiveness of the trojans. Next, Section 5.2 examines the overhead introduced by the trojans. Finally, Section 5.3 compares the trojans with other state-of-the-art trojans.

### 5.1 IMPACT OF TROJAN ATTACKS ON ACCURACY

This section will present the functional results after the accelerator was infected by the two different trojans with three targets each.

#### 5.1.1 *Gradually Degrading Accuracy Trojan*

The **GDATs** functional impact is detailed in Table 9. The baseline accuracy of the **RTL** model was 98.697%. When infected by a trojan that targets a weight in the first convolutional layer, the accuracy after the functional testing period was 98.597%. It would take 74 days for the trojan to reach its desired value, which would lead to an accuracy of 22.144% in the long term. When attacking a bias in the second fully connected layer, the accuracy after the functional testing period was 98.297%; This would take 47.3 days to reach its desired value, resulting in an accuracy of 35.371% in the long term. For the feature map attack, it had a 98.096% accuracy after the functional testing period and 83.3 days to reach its desired value, leading to an accuracy of 20.541% long term.

Table 9: Accuracy of the GDATs compared to baseline.

	Baseline	Weight Trojan	Bias Trojan	Feature Map Trojan
Accuracy after 2 weeks	98.697%	98.597%	98.297%	98.096%
Time to reach max effect	-	74 days	47,3 days	83,3 days
Long term accuracy	-	22.144%	35.371%	20.541%

#### 5.1.2 *Suddenly Degrading Accuracy Trojan*

The **SDAT** functional impact is shown in Table 10. Baseline accuracy stays the same, but compared to its incrementing counterpart, the **SDAT** has no effect on the accuracy after two weeks, since it is designed to activate after six months and therefore has not yet been

triggered. The long-term accuracy of both trojans also stays the same since they have the same payloads.

Table 10: Accuracy of the SDAT compared to baseline.

	Baseline	Weight Trojan	Bias Trojan	Feature Map Trojan
Accuracy after 2 weeks	98.697%	98.697%	98.697%	98.697%
Time to reach max effect	-	6 months	6 months	6 months
Long term accuracy	-	22.144%	35.371%	20.541%

## 5.2 RESOURCE UTILIZATION AND OVERHEAD

This section will present the overhead introduced by the different versions and the trojan. In Section 5.2.1, the hardware overhead by the different versions of the trojans is shown. Section 5.2.2 presents the power overhead introduced by the trojans.

### 5.2.1 Hardware Overhead

Figure 12 and 13 showcases the resource utilization before and after the different trojans were injected.

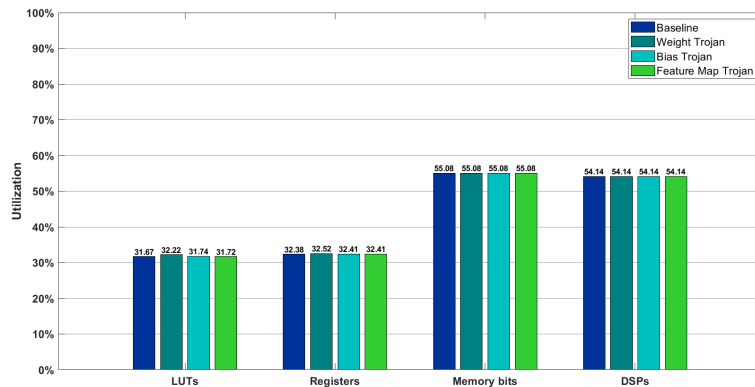


Figure 12: GDATs hardware utilization of total available hardware.

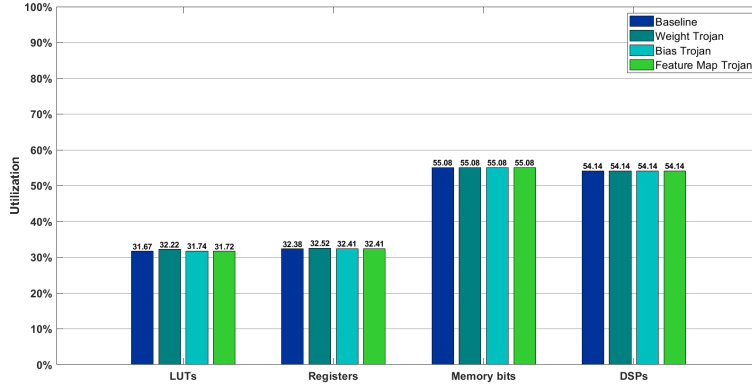


Figure 13: SDATs hardware utilization of total available hardware.

Since the memory and Digital Signal Processors (DSP) usage were not increased by any of the trojans, the following Tables will exclude those as a measurement. Table 11 showcases the hardware usage of a CNN containing the GDATs.

Table 11: Hardware utilization of accelerator with GDATs.

	Baseline	Weight Trojan	Bias Trojan	Feature Map Trojan
LUTs	36260	36958	36379	36365
Registers	37071	37228	37103	37102
LUTs Increase	-	1.92%	0.33%	0.29%
Registers Increase	-	0.42%	0.09%	0.08%

Table 12 showcases the hardware usage of a CNN containing the SDATs.

Table 12: Hardware utilization of accelerator with SDATs.

	Baseline	Weight Trojan	Bias Trojan	Feature Map Trojan
LUTs	36260	36883	36333	36316
Registers	37071	37227	37101	37100
LUTs Increase	-	1.72%	0.20%	0.15%
Registers Increase	-	0.42%	0.08%	0.08%

### 5.2.2 Power Consumption Overhead

Table 13 showcases the power consumption of a CNN containing the trojans as estimated by Intel's Early Power Estimator for a Cyclone IV running at 125 MHz [13].

Table 13: Power estimation of accelerator with GDATs.

	Baseline	Weight Trojan	Bias Trojan	Feature Map Trojan
Power usage (W)	1.237	1.248	1.239	1.239
Difference (W)	-	0.011	0.002	0.002
Increase	-	0.89%	0.16%	0.16%

Table 14 showcases the estimated power consumption usage of a CNN containing the SDAT.

Table 14: Power estimation of accelerator with SDATs.

	Baseline	Weight Trojan	Bias Trojan	Feature Map Trojan
Power usage (W)	1.237	1.247	1.239	1.239
Difference (W)	-	0.010	0.002	0.002
Increase	-	0.81%	0.16%	0.16%

### 5.3 COMPARATIVE ANALYSIS WITH EXISTING TROJANS

Table 15 presents the hardware overhead results of the implemented trojans compared to two state-of-the-art trojans sorted in LUT overhead. The first attack from Hou et al. [11] is a data-based triggered trojan that attacks the reconfigurable network; introduced a 0.24% overhead in LUT utilization while not providing any data on register utilization. The second comparison involves a time-based trojan Int-Monitor [19], which, when synthesized on a ZCU102 board, resulted in a 0.37% LUT overhead and 0.03% register overhead.

The trojan with the least hardware overhead, *Feature Map SDAT*, increased LUT utilization by 0.15% and register utilization by 0.08%. The trojan with the most hardware overhead, *Weight GDAT*, recorded a 1.92% LUT overhead and a 0.42% register overhead. Like the Int-Monitor trojan, our implementations did not increase BRAM and DSP utilization

Table 15: Hardware utilization of HTs.

Trojans	LUT Overhead	Register Overhead
Feature Map SDAT	0.15%	0.08%
Bias SDAT	0.20%	0.08%
Hou et al. [11]	0.24%	-
Feature Map GDAT	0.29%	0.08%
Bias GDAT	0.33%	0.09%
Int-Monitor NVDLA [19]	0.37%	0.03%
Weight SDAT	1.72%	0.42%
Weight GDAT	1.92%	0.42%

In Table 16, the power consumption overhead of the implemented trojans with that of the Int-Monitor trojan [19] is listed. Both the GDAT and SDAT trojans, when targeting bias and feature maps, introduced a power consumption increase of 0.16% compared to the baseline. These trojans outperformed the Int-Monitor trojan in terms of power overhead. However, the trojans that targeted the weights incurred significantly higher power overheads, at 0.81% and 0.89%, respectively.

Table 16: Power consumption overhead.

Trojans	Power overhead
Feature Map SDAT	0.160%
Bias SDAT	0.160%
Feature Map GDAT	0.160%
Bias GDAT	0.160%
Int-Monitor NVDLA [19]	0.187%
Weight SDAT	0.810%
Weight GDAT	0.890%



## DISCUSSION

---

This chapter provides an in-depth discussion of the findings and their implications. Section 6.1 offers an analysis of the results presented in the previous chapters. In Section 6.2, we explore the practical realization of these attacks and their potential impacts. Section 6.3 outlines potential directions for future research. Finally, Section 6.4 revisits the research questions of this thesis.

### 6.1 DISCUSSION OF RESULTS

Among our implementations, the best-performing trojan in terms of hardware overhead was the feature map *SDAT*. It demonstrated more efficient *LUT* usage compared to the two other trojans in our study. It increased *LUT* usage by only 0.15%, which is favorable compared to the 0.24% increase observed with *Hou et al.*'s trojan and the 0.37% increase by *Int-Monitor*. The efficient *LUT* usage is not entirely unexpected since a time-based trigger design requires very little hardware to be realized. They rely solely on counters and comparators, unlike data-based triggers, which may necessitate more complex logic units. Although it is simpler, this does not imply it is worse. A time-based trigger has the advantages of being very small and not requiring an external trigger.

However, all of our trojans incurred a higher register overhead compared to the *Int-Monitor* trojan. The *SDATs* required a large counter, enabling the trojan to remain dormant for an extended period, while the *GDAT* used a smaller counter but required additional registers to store the payloads.

When comparing our implementations, the increased complexity of the *GDATs* compared to the *SDATs* led to greater *LUT* overhead, as it required additional hardware to increase the payload applied to the target periodically. Among our implementations, the best-performing *GDAT* was the one inserted into the feature maps, closely followed by the bias *GDAT*. When synthesizing a design, a lot of optimization is done with performance, power, and area in mind. Therefore, these components likely had the most underutilized *LUTs*, enabling the synthesis optimization to better conceal these trojans in existing hardware, compared to the trojan targeting the weights. The results clearly show that the trojan implemented within the weights had the most hardware overhead.

Similar to hardware overhead, the power consumption increase for both *SDAT* and *GDAT* attacks on biases and feature maps was lower

than that of the Int-Monitor trojan. According to [Li and Hou](#), "the Int-Monitor's power overhead is so negligible that it could rarely be detected by defense technologies such as on-chip sensors." Our trojans further improve upon this result. However, for the trojans attacking the weights, the power consumption significantly increased, most likely related to the increase in hardware needed for these trojans.

Regarding attack effectiveness, we were able to significantly reduce accuracies with all attacks by targeting one of the few weights, biases, or feature maps identified as vulnerable by the sensitivity analysis. This finding aligns with studies on pruning in neural network architectures like LeNet-5, which found that a lot of neurons and weights contributed very little to the final classification and could be removed [10].

By offering multiple potential targets and payloads, the attacks can degrade the accuracy anywhere from 1% to 78.16%, based on how noticeable the attack is intended to be when fully executed. For instance, an accuracy degradation of only 5% might go unnoticed during inference, depending on the application of the accelerator. However, if the accuracy is significantly degraded, it is likely to eventually be detected during inference, potentially leading to product recalls and damaging the reputation of companies distributing the accelerators.

## 6.2 FEASIBILITY OF ATTACK

Considering the potential real-world implications of this attack, including economic damage, erosion of trust, and even safety hazards depending on the application, it's important to discuss the feasibility of the attack and what version of the trojan to employ.

While it may seem improbable for reputable companies like Intel or AMD to have compromised CAD software, the possibility cannot be dismissed outright and is one of the attack vectors discussed by both [Li et al.](#) and [Zhang and Qu](#) [18, 33]. The CAD tools are incredibly powerful, and if you can't trust them, you are severely limited. For example you can't really create a golden sample of your design, since you need the CAD tools to create it. The only alternative would be designing the component using different CAD tools. However, any subtle variations in hardware or power consumption could easily be attributed to differing optimization between tools or the diverse platforms for which the design is synthesized. Furthermore, the trojan could be introduced by a rogue designer and does not necessarily need to originate from the synthesis tools.

Taking into account these potential attack vectors and the assertion by [Li and Hou](#) that the best trojans should go unnoticed due to its overhead being less than 1% [19], there is a possibility that these trojans could find its way into a product and have real consequences.

What sets this attack apart is that unless you marginally reduce accuracy, it's not necessarily designed to remain undetected indefinitely. Rather, it aims to evade detection during functional testing and for a period during field inference. Some of the damage it inflicts actually occurs after detection, particularly when the product requires recall. Since this novel attack focuses on undermining trust and causing economic harm, its ultimate objective is distinct and can't be compared to other trojans.

Comparing the two versions of our trojan, the [SDAT](#) is superior in both terms of hardware efficiency and power consumption. Furthermore, it trivializes passing functional testing. However, the choice between the sudden and gradual version of the trojan depends on the specific scenario, as they are not functionally identical. Although both eventually lead to the same long-term outcome, their intermediate stages differ significantly. With [SDAT](#), everything appears normal until a sudden drop in accuracy occurs. On the other hand, the [GDAT](#) operates stealthily in the background, slowly diminishing accuracy over time. This prolonged intermediate stage, which can vary depending on the time to activation of the implemented trojan, can cause certain damages that the sudden trojan version cannot. The gradual degradation may not be immediately noticeable, and consequently, there will be a period where users receive and potentially trust less accurate predictions. For instance, in automated license plate identification, a gradual degradation will lead to a slight increase in misclassifications, necessitating more human intervention and correction. This incurs a gradual increase in need for additional man hours and financial costs during the intermediate stage, until inevitably suspicion grows and the attack gets detected. This kind of attack would not be possible with the sudden and abrupt change in performance the [SDAT](#) would produce.

Therefore, while the sudden trojan version may offer advantages in terms of efficiency and simplicity, the gradual version can be a strategic choice in situations where subtle, incremental changes are preferred over abrupt disruptions.

It's worth discussing the sensitivity analysis due to its demanding and time-consuming process. Although an attack could theoretically proceed without it, the sensitivity analysis plays a vital part in highlighting critical vulnerabilities within the network. Without such analysis, the effects of an attack targeting random weights or biases become unpredictable. Additionally, the results of the sensitivity analysis show that only a small fraction of weights significantly impact overall accuracy when manipulated. Specifically, just 0.15% of all weights in the network degrade accuracy by over 5%.

If a target were to be randomly selected, once the bitstream file is generated, the attack target is effectively locked, meaning the same infected bitstream file is uploaded to all [FPGAs](#) expecting the same

design. If the randomly chosen target does not impact accuracy, all [FPGAs](#) will receive an ineffective attack and a new target can only be chosen the next time a bitstream file is generated from the CAD tool.

Consequently, an attack that randomly targets an individual weight across the network is unlikely to lead to significant accuracy degradation and has thus not been studied.

### 6.3 FUTURE WORK

This section explores potential directions for future research, focusing on both detectability and the expansion of the attack.

#### 6.3.1 *Stealthiness to trigger detection methods*

While out of this project's scope, an extensive evaluation of these trojans' stealthiness against current trigger detection methods is needed. A paper by [Trippel et al.](#) introduced a trigger detection program. They argue that "one-size-fits-all approaches are incomplete and akin to proving a design is bug-free." Instead of attempting to verify that a design is free of all types of trojans, they advocate for a divide-and-conquer approach. Consequently, this defense strategy is specifically tailored to identify timer-based triggers. Their defense mechanism can be implemented at the [RTL](#) stage or after synthesis. This implies that it has the capability to detect our attack, whether it was injected by the synthesis tool as intended or by a rogue designer since there is nothing about this attack that specifically requires the attack vector to be the CAD tools. The paper outlines two defining properties of a timer-based trigger they look for [25].

- **Property 1:** The timer-based trigger does NOT repeat a value without a system reset.
- **Property 2:** The timer-based trigger does NOT enumerate all possible values without activating.

Drawing from these two properties, their defense mechanism should theoretically be capable of detecting the [SDAT](#) trojan but may struggle to identify the incrementing variant. The [GDAT](#) triggers repeatedly to increase its payload. According to property 1, this defense would likely fail to flag the [GDAT](#) because each trigger event resets the timed trigger, causing it to repeat values without a system reset, potentially making it indistinguishable from a normal loop behaviorally. However, these conclusions are speculative, based on the properties defined by the authors. The paper also discusses various other defense strategies aimed at identifying timer-based triggers, which could serve as a valuable starting point for future research into understanding the stealthiness of trojans against current detection methodologies.

### 6.3.2 Attacking other model architectures

The LeNet-5 model used in this thesis, relatively speaking, is a small CNN with only 79,242 parameters. How sensitive a larger CNN architecture like Alexnet, which has 62.3 million parameters [23], would be to this kind of an attack could be of interest to research.

### 6.3.3 Intentionally overflowing Trojan

One intriguing observation made was that despite extensive efforts to determine the maximum payload that would fit within the Q1.14 format without overflowing, there were instances where overflowing actually resulted in greater accuracy degradation compared to not overflowing. While the unpredictability can be a significant drawback, if an attacker desired even further accuracy degradation, additional exploration could involve leveraging overflowing to achieve this. We conducted some preliminary experiments deliberately causing overflow in neuron 10 (number 9) of the last fully connected layer by selecting a bias trojan payload large enough to push the output above 1.9999 when the neuron output naturally tended to be high (indicating a correct prediction). In contrast, this attack would also increase the output value in cases where the natural output tended to be low. Basically, what this attack tries to achieve is predicting the number 9 every time, except the times the number 9 is correct, in which case it overflows. When testing it, we were able to achieve a long-term accuracy of 7.69%. Figure 14 showcases the output values of neuron 10 with the idea of pushing correct predictions over the overflow threshold while only increasing the rest.

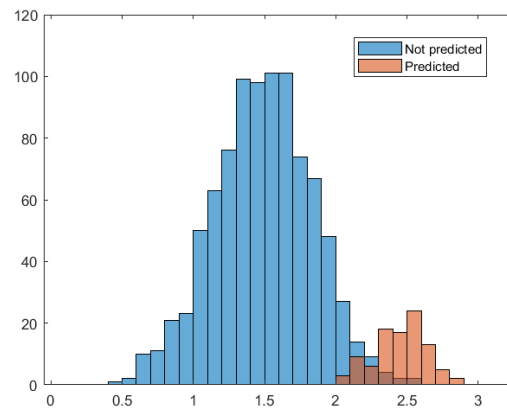


Figure 14: Histogram of prediction values of neuron 10.

Exploring the feasibility of this approach and extending it to experiment with overflowing layers other than the output layer could be worth further investigation.

#### 6.4 ADDRESSING RESEARCH QUESTIONS

*From an adversary's perspective, which components of a neural network accelerator are susceptible to an accuracy-degrading hardware trojan that deteriorates accuracy over time?*

An attacker aiming to maximize the effectiveness of an attack (i.e., degrade accuracy as much as possible) would focus on targets identified as particularly susceptible through the sensitivity analysis. As the attacker is also interested in introducing minimal additional hardware, they would focus on the target with the most underutilized LUTs and registers in order to hide their presence better. In the case of this particular design this would be targeting the memory components for the feature maps and biases.

*In real-world scenarios, how well would an accuracy-degrading hardware trojan be hidden compared to other types of hardware trojans targeting neural network accelerators?*

Since the trojans used in the comparison in this thesis have different objectives compared to ours, it is not possible to directly compare their stealthiness. However, from a hardware perspective, the accuracy-degrading trojans introduced in this study do not add significant overhead. In fact, they often introduce similar or even less overhead compared to the other trojans.

## CONCLUSION

---

This study presents a new type of attack that can degrade the accuracy of a **CNN** accelerator over time. Two variants (**GDAT** and **SDAT**) of this attack were injected into three components of a **CNN** accelerator to evaluate their effectiveness from the perspectives of stealthiness, hardware overhead, and attack effectiveness. All of the attacks managed to stay stealthy throughout functional testing, having a minimal impact on accuracy during this period. The attacks implemented in feature maps induced the lowest amount of overhead, closely followed by the biases. The attack effectiveness of all attacks was high. However, attacking feature maps resulted in the most significant degradation of accuracy, potentially reducing it to as low as 20.5%, as seen in Table 9. Despite these trojans remaining hidden in terms of hardware overhead, further investigation is needed to evaluate their resilience against state-of-the-art trigger detection methods. Additionally, exploring the potential expansion of this attack to other model architectures could provide deeper insights into its versatility and impact.

In conclusion, this type of attack presents a realistic threat with applications for both types of trojans. Attackers can customize the trojan to suit their objectives, providing a dynamic approach where they can select the target, timing of activation, and whether to implement gradual or sudden degradation, ultimately determining the long-term impact on accuracy.



## BIBLIOGRAPHY

---

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. Accessed: 2024-04-10.
- [2] Rajat Subhra Chakraborty, Seetharam Narasimhan, and Swarup Bhunia. Hardware trojan: Threats and emerging solutions. In *2009 IEEE International High Level Design Validation and Test Workshop*, pages 166–171, 2009. doi: 10.1109/HLDVT.2009.5340158.
- [3] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. 2018. doi: 10.48550/arXiv.1811.03728.
- [4] Joseph Clements and Yingjie Lao. Hardware trojan attacks on neural networks. *CoRR*, abs/1806.05768, 2018. URL <http://arxiv.org/abs/1806.05768>.
- [5] Joseph Clements and Yingjie Lao. Backdoor attacks on neural network operations. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1154–1158, 2018. doi: 10.1109/GlobalSIP.2018.8646335.
- [6] Joseph Clements and Yingjie Lao. Hardware trojan design on neural networks. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2019. doi: 10.1109/ISCAS.2019.8702493.
- [7] Chen Dong, Yi Xu, Ximeng Liu, Fan Zhang, Guorong He, and Yuzhong Chen. Hardware trojans in chips: A survey for detection and prevention. *Sensors*, 20(18), 2020. ISSN 1424-8220. doi: 10.3390/s20185165. URL <https://www.mdpi.com/1424-8220/20/18/5165>.

- [8] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Jincheng Yu, Junbin Wang, Song Yao, Song Han, Yu Wang, and Huazhong Yang. Angel-eye: A complete design flow for mapping cnn onto embedded fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):35–47, 2018. doi: 10.1109/TCAD.2017.2705069.
- [9] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. [dl] a survey of fpga-based neural network inference accelerators. *ACM Trans. Reconfigurable Technol. Syst.*, 12(1), mar 2019. ISSN 1936-7406. doi: 10.1145/3289185. URL <https://doi.org/10.1145/3289185>.
- [10] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. 2015. doi: 10.48550/arXiv.1506.02626.
- [11] Jia Hou, Zichu Liu, Zepeng Yang, and Chen Yang. Hardware trojan attacks on the reconfigurable interconnections of field-programmable gate array-based convolutional neural network accelerators and a physically unclonable function-based countermeasure detection technique. *Micromachines*, 15(1), 2024. ISSN 2072-666X. doi: 10.3390/mi15010149. URL <https://www.mdpi.com/2072-666X/15/1/149>.
- [12] Intel. Cyclone<sup>®</sup> iv ep4CE115, 2009. URL <https://www.intel.com/content/www/us/en/products/sku/210467/cyclone-iv-ep4ce115-fpga/specifications.html>. Accessed: 2024-02-28.
- [13] Intel. Cyclone<sup>®</sup> iv and cyclone<sup>®</sup> v powerplay early power estimator, 2017. URL <https://www.intel.com/content/www/us/en/support/programmable/support-resources/power/cy4-5-estimator-download.html>. Accessed: 2024-05-02.
- [14] Intel. Quartus prime development software, 2024. URL <https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html>. Accessed: 2024-05-02.
- [15] Sara Kaviani and Insoo Sohn. Defense against neural trojan attacks: A survey. *Neurocomputing*, 423:651–667, 2021. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2020.07.133>. URL <https://www.sciencedirect.com/science/article/pii/S0925231220316350>.
- [16] Christian Krieg, Clifford Wolf, and Axel Jantsch. Malicious lut: A stealthy fpga trojan injected and triggered by the design flow. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE Press, 2016. doi: 10.1145/2966986.2967054. URL <https://doi.org/10.1145/2966986.2967054>.

- [17] Sen M. Kuo and Woon-Seng Gan. *Digital Signal Processors: Architectures, Implementations, And Applications*. Prentice Hall, 2004. ISBN 978-0130352149.
- [18] He Li, Qiang Liu, and Jiliang Zhang. A survey of hardware trojan threat and defense. *Integration*, 55:426–437, 2016. ISSN 0167-9260. doi: <https://doi.org/10.1016/j.vlsi.2016.01.004>. URL <https://www.sciencedirect.com/science/article/pii/S0167926016000067>.
- [19] Peng Li and Rui Hou. Int-monitor: A model triggered hardware trojan in deep learning accelerators. *The Journal of Supercomputing*, 79(3):3095–3111, 2 2022. doi: 10.1007/s11227-022-04759-y. URL <https://doi.org/10.1007/s11227-022-04759-y>.
- [20] Yuntao Liu, Ankit Mondal, Abhishek Chakraborty, Michael Zuzak, Nina Jacobsen, Daniel Xing, and Ankur Srivastava. A survey on neural trojans. In *2020 21st International Symposium on Quality Electronic Design (ISQED)*, pages 33–39, 2020. doi: 10.1109/ISQED48828.2020.9137011.
- [21] Erick L. Oberstar. Fixed-point representation fractional math, 2007. URL <https://web.archive.org/web/20171104111827/http://www.superkits.net/whitepapers/Fixed%20Point%20Representation%20%26%20Fractional%20Math.pdf>. Accessed: 2024-03-19.
- [22] Jordan Robertson and Michael Riley. The big hack: How china used a tiny chip to infiltrate america’s top companies, 10 2018. URL <https://gssd.mit.edu/search-gssd/site/big-hack-how-china-used-tiny-chip-61791-mon-05-06-2019-2051>. Accessed: 2024-04-10.
- [23] Shipra Saxena. Introduction to the architecture of alexnet, 2023. URL <https://www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/>. Accessed: 2024-05-8.
- [24] Siemens. Hdl simulation modelsim, 2019. URL <https://eda.sw.siemens.com/en-US/ic/modelsim/>. Accessed: 2024-01-10.
- [25] Timothy Trippel, Kang G. Shin, Kevin B. Bush, and Matthew Hicks. Bomberman: Defining and defeating hardware ticking timebombs at design-time. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 970–986, 2021. doi: 10.1109/SP40001.2021.00052.
- [26] Ashutosh Kumar Upadhyay. Convolution neural network - simple code - simple to use, 2017. URL <https://se.mathworks.com/matlabcentral/fileexchange/59223-convolution-neural-network-simple-code-simple-to-use>. Accessed: 2024-02-13.

- [27] Adam Waksman and Simha Sethumadhavan. Tamper evident microprocessors. In *2010 IEEE Symposium on Security and Privacy*, pages 173–188, 2010. doi: 10.1109/SP.2010.19.
- [28] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723, 2019. doi: 10.1109/SP.2019.00031.
- [29] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A. Gunter, and Bo Li. Detecting ai trojans using meta neural analysis, 2020.
- [30] Chen Yang, Yizhou Wang, Xiaoli Wang, and Li Geng. Wra: A 2.2-to-6.3 tops highly unified dynamically reconfigurable accelerator using a novel winograd decomposition algorithm for convolutional neural networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(9):3480–3493, 2019. doi: 10.1109/TCSI.2019.2928682.
- [31] Jing Ye, Yu Hu, and Xiaowei Li. Hardware trojan in fpga cnn accelerator. In *2018 IEEE 27th Asian Test Symposium (ATS)*, pages 68–73, 2018. doi: 10.1109/ATS.2018.00024.
- [32] Grant Yu. Mnist\_classification\_fpga, 2019. URL [https://github.com/grant4001/MNIST\\_Classification\\_FPGA](https://github.com/grant4001/MNIST_Classification_FPGA). Accessed: 2024-02-13.
- [33] Jiliang Zhang and Gang Qu. A survey on security and trust of fpga-based systems. In *2014 International Conference on Field-Programmable Technology (FPT)*, pages 147–152, 2014. doi: 10.1109/FPT.2014.7082768.





PO Box 823, SE-301 18 Halmstad  
Phone: +35 46 16 71 00  
E-mail: [registrator@hh.se](mailto:registrator@hh.se)  
[www.hh.se](http://www.hh.se)