# ENHANCED HARDWARE TROJAN DETECTION IN CHIPS BY REDUCING LINEARITY BETWEEN FEATURES


by

Alfred Moussa


A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Electrical and Computer Engineering

Boise State University

August 2023

BOISE STATE UNIVERSITY GRADUATE COLLEGE

**DEFENSE COMMITTEE AND FINAL READING APPROVALS**

of the thesis submitted by

Alfred Moussa

Thesis Title: Enhanced Hardware Trojan Detection in Chips By Reducing Linearity Between Features

Date of Final Oral Examination: 28 April 2023

The following individuals read and discussed the thesis submitted by student Alfred Moussa, and they evaluated the student's presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

| | |
|---|---|
| Nader Rafla Ph.D. | Chair, Supervisory Committee |
| Benjamin Johnson Ph.D. | Member, Supervisory Committee |
| Jennifer Smith Ph.D. | Member, Supervisory Committee |

The final reading approval of the thesis was granted by Nader Rafla Ph.D., Chair of the Supervisory Committee. The thesis was approved by the Graduate College.

# DEDICATION

This thesis is dedicated to my mother who encouraged me to pursue my education and attain a master's degree. Without the unwavering love and support of my parents, and their constant push for improvement, I would not have accomplished anything significant.

# ACKNOWLEDGMENT

I would like to express my deepest gratitude to Dr.Nader Rafla, for his invaluable guidance, support, and mentorship throughout this journey. His unwavering commitment to excellence, keen insights, and constructive feedback has been instrumental in shaping the direction of my research and the development of my academic skills.

I would also like to extend my sincere thanks to my professors, whose teachings and expertise have enriched my academic experience and contributed to the completion of this thesis. Their knowledge, dedication, and enthusiasm for their respective fields have inspired me to push my boundaries and pursue my academic goals with passion and commitment.

Finally, I would like to express my deepest appreciation to my family, whose unwavering love, support, and encouragement have been a constant source of motivation and inspiration.

Thank you all for your contributions, guidance, and unwavering support. Without your help, this accomplishment would not have been possible.

# ABSTRACT

Globally, there has been an increase in demand for System on Chip (SoC) applications, active medical implants, and Internet of Things (IoT) devices. However, due to challenges in the global supply chain, the design, fabrication, and testing of Integrated Circuits are often outsourced to untrusted third-party entities around the world rather than a single trusted entity. This situation presents an opportunity for adversaries to compromise the device's integrity, performance, and functionality by inserting malicious modifications known as Hardware Trojans (HTs) into the original design. HTs can also create a backdoor in the system for malicious alterations.

The problem of hardware trojans is tackled in this thesis through the application of two types of machine learning models. The proposed methodology involves utilizing netlist features of the digital hardware design generated from synthesis and inputting them into the machine learning model. Additionally, measures are taken to prevent interdependence among features, which could lead to overfitting on the training dataset.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS

**3PIP** Third Party IPs

**HDLs** Hardware description languages

**HT** Hardware Trojan

**ICs** Integrated circuits

**IoT** Internet of Things

**ML** Machine learning

**RFID** radio frequency identification devices

**SoC** system-on-a-chip

# CHAPTER 1:
# INTRODUCTION

Internet of Things (IoT) is a huge network that encompasses and connects various information-sensing devices, such as: radio frequency identification devices (RFID), infrared sensors, and global positioning systems. The IoT is emerging as the third wave of the world information industry after computers and the internet. The prevalence of Integrated circuits (ICs) increases in various fields in IoT, and the potential risk of hardware Trojans in ICs are becoming a significant concerns, as they are used daily and must store and process huge amounts of user-sensitive data. If an attacker exploits such information, it may seriously damage or endanger user privacy, which can lead to threatening users' safety.

The core component of an IoT device is a system-on-a-chip (SoC). However, due to the current trend of decreasing size and increasing complexity, modern SoC designs have become significantly challenging and expensive for chip manufacturers to handle. To address this, manufacturers are turning to Third Party IPs (3PIP) which provides cost-effective solutions due to the reusability of IP cores. This allows manufacturers to allocate their resources to meet market demands and pressure to meet deadlines. However, the security and reliability of 3PIP cannot be guaranteed, and relying on untrusted IPs can greatly increase the risk of hardware trojan insertion.

A Hardware Trojan (HT) is a harmful modification made to an Integrated Circuit that can result in the leakage of sensitive information, system functionality changes, performance degradation, denial-of-service (DoS), or even create a backdoor to the entire system. Given the widespread use of IC chips in critical infrastructure, military devices, and biomedical devices, an undetected Hardware Trojan could be fatal.

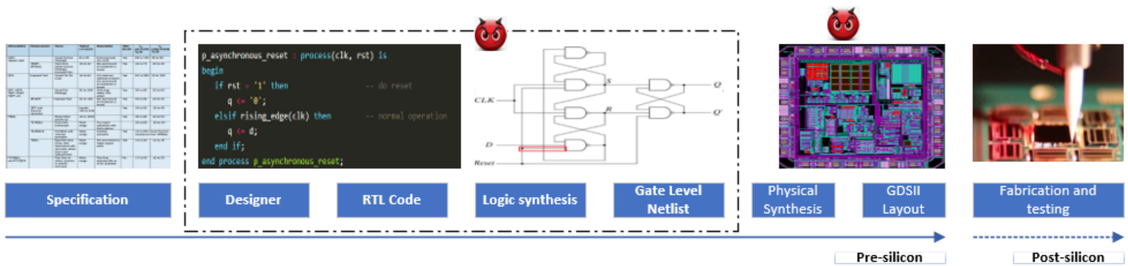


**Figure 1.1: IC development phase.**

The standard IC physical design and layout process, depicted in Figure 1.1, spans from system-level specification to fabrication and testing. At each stage of this process, there exists the possibility of incorporating a Hardware Trojan into the IC. The objective of this study is to identify any HT that may have been inserted during the design phase.

Suppose a company has sent its physical digital hardware design known as "Graphic Design System II (GDSII)" to a fabrication company that may not be trustworthy. A recent study by Rajarathnam et al [1] introduces a reverse engineering framework named ReGDS. ReGDS employs a technology library to extract transistor-level connectivity information from the GDSII layout, and utilizes relationship-based matching to identify logic gates and subsequently retrieve the original gate-level netlist. The outcome of the research revealed that ReGDS succeeded in recovering the original digital design from the layout with a 100% success rate.

To prevent this, I propose that the design company should verify the authenticity of the manufactured chip by measuring specific features of the fabricated chip, such as power consumption, power leakage, and area. This can be done using techniques such as Scanning Electron Microscopy (SEM) or Focused Ion Beam (FIB) to measure the chip's layers and area. These features are crucial in digital hardware design because power and area have a relationship.

In the event that the fabrication company inserts a logic trojan that remains dormant until a particular trigger is activated, it could conceal any surge in power consumption by having a Logic Gate that does not consume power until the trigger is activated. This would usually result in an increase in the overall area or the number of layers of the chip. By inputting these features into a machine-learning model before and after fabrication, linearity between features can be checked to avoid overfitting and ensure a high probability of detecting any malicious modifications introduced by the untrustworthy fabrication company.

## 1.1 Characterization of Hardware Trojans

A Hardware Trojan consists of a trigger and a payload circuit. The trigger mechanism monitors the chip and activates the payload under rare conditions to evade possible HT detection solutions in the post-fabrication testing, as illustrated in Figure 1.2.

HTs pose several threats to the security of electronic devices such as:

1. Data theft: HTs can be designed to leak data, such as encryption keys, user data, or confidential information, to an attacker.

2. Denial of service: HTs can be designed to cause the device to fail, disrupt the system's normal functioning, or cause it to shut down, leading to a denial of service attack.

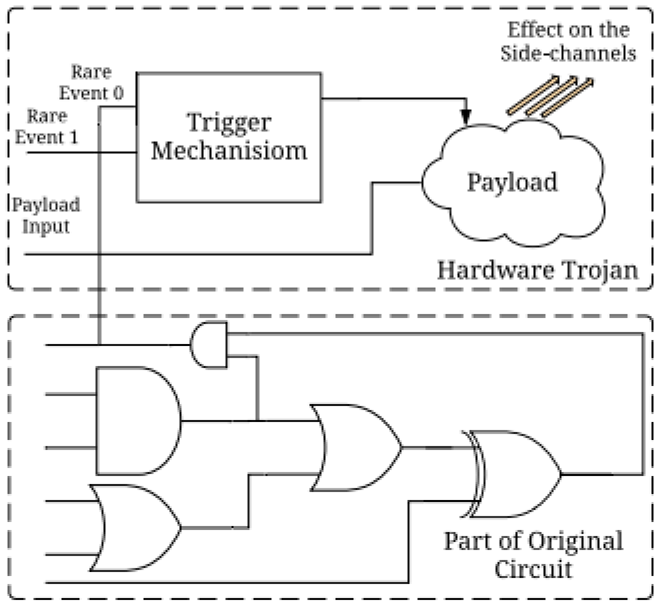


**Figure 1.2: HT with a trigger mechanism and a payload [2]**

3. Malware insertion: HTs can be designed to allow attackers to insert malware onto the device, which can then be used to gain access to other parts of the network.

4. Backdoors: HTs can be designed to create backdoors in the device's security, allowing attackers to bypass security measures and gain unauthorized access.

## 1.2 Effect of Hardware Trojans

Hardware Trojans are malicious modifications made to the hardware of a chip during its design or manufacturing process, which can cause a wide range of detrimental effects. The exact impact of a hardware Trojan depends on its design and goals.

For instance, a suspected nuclear facility in Syria was bombed by Israeli jets in 2007, because Syrian radar was crippled by a remote kill switch thru a backdoor in its commercial off-the-shelf microprocessor [3]. Similarly, the U.S. military in 2010 discovered a hardware Trojan embedded in over 59,000 microchips purchased for use in various systems, ranging from missile defense to friend-or-foe identification devices. This hardware Trojan gave adversaries a backdoor entry into their entire system [4]. In 2012, HTs were found in Actel/Microsemi ProA-SIC3 chips, which were used in military-grade FPGAs, and they added unwanted JTAG functionality to the silicon itself, allowing adversaries to extract secret keys, manipulate the chip's configuration, and take control of the system [5]. Furthermore, after observing unusual network activity and firmware problems in 2015, Apple detected a questionable chip in their Supermicro servers [6].

## 1.3 Research Motivation

The presence of these Trojans on a chip can lead to data theft, device malfunction, and other dangerous consequences. As the complexity and diversity of ICs continue to grow, traditional techniques for detecting Hardware Trojans such as manual inspection, side-channel analysis, and fault injection become less effective and efficient.

To address this challenge, researchers have turned to Machine Learning techniques to detect Hardware Trojans on chips. Machine Learning models can learn from a large dataset of ICs with and without Trojans and identify patterns and anomalies in the circuit behavior. These models can then be used to automatically detect the presence of Hardware Trojans in new ICs.

Machine learning (ML) has emerged as a significant tool in detecting hardware trojans, which can enhance the security and reliability of integrated circuits. In several critical applications, such as automotive, aerospace, and defense systems, the integrity of these circuits is crucial. By utilizing ML algorithms, it is possible to improve the detection of hardware trojans and ensure the trustworthiness of integrated circuits.

## 1.4 Thesis Statement

The objective of this research is to answer the following question: **In the pre-silicon phase, is there a reliable and efficient method to detect the presence of a trojan that may have been embedded in the digital hardware design?**

## 1.5 Contributions and outline

The key contribution of this research is on two aspects:

1. The feature extraction from digital hardware design after synthesis.
2. The reduction of linearity between features that are caused by scaling the netlist features in ML.

The reason for reducing linearity between features is that machine learning algorithms cannot comprehend measurement units such as power (mW) or area ($um^2$), and can only process numerical values. Therefore, each feature is scaled between (0,1), resulting in linearity between features. Reducing these linearities can help the machine learning algorithm avoid overfitting the training data, leading to improved performance for hardware trojan detection across different trust benchmarks [7].

The rest of the thesis is organized as follows: Chapter 2 of the thesis provides a comprehensive background on Hardware Trojan, Machine Learning, and Literature work. In Chapter 3, the extracted features and algorithm selection for detecting hardware trojans are presented. Chapter 4 presents the efficiency and accuracy of each machine learning model using the Confusion matrix. Lastly, chapter 5 presents the conclusions derived from the research and future work.

# CHAPTER 2:

# BACKGROUND

## 2.1 Introduction to Machine Learning

Machine learning is a sub-field of artificial intelligence that focuses on developing algorithms and models that enable computers to learn from data and make predictions or decisions based on that learning. The goal of machine learning is to develop computational methods that can improve automatically through experience, without being explicitly programmed.

In machine learning, the algorithms are trained on a dataset, which contains input data and corresponding output data, known as labels. The algorithms use this data to learn patterns, relationships, and correlations in the data, and to build models that can predict the output for new, unseen data. The models are then tested on a separate dataset, called the test set, to evaluate their accuracy and generalization performance.

There are different types of machine learning techniques, including supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. Supervised learning involves training the algorithms on labeled data, while unsupervised learning involves training the algorithms on unlabeled data. Semi-supervised learning uses a combination of labeled and unlabeled data, while reinforcement learning involves training algorithms to make decisions in an environment to maximize a reward signal. In this work, it uses the supervised and unsupervised machine learning to extract insights and knowledge from data.

### 2.1.1 Supervised Machine Learning

Supervised learning is a type of machine learning where the algorithms are trained on labeled data, meaning that the input data has a known output or target variable. The goal of supervised learning is to learn a mapping function from the input variables to the output variable. This mapping function can then be used to make predictions on new, unseen data. There are two main types of supervised learning: classification and regression.

In Figure 2.1, on the left it shows a dataset were a classification is required to effectively divide the dataset into classes based on different parameters; on the right, it shows correlations between dependent and independent variables in order to predict the continuous variables such as prediction of Market Trends, or one of the most common example is to predict student passing or failing exams according to the combination of number of hours he/she has slept and hours spent for studying.

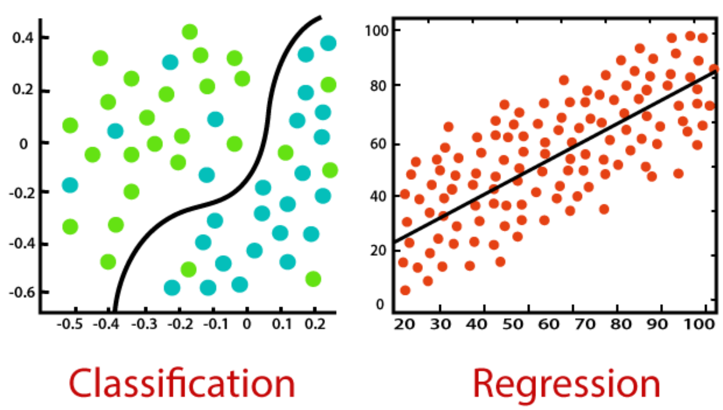


**Figure 2.1: Classification vs. Regression**[8]

#### 2.1.1.1 Classification

The algorithm learns to predict the class or category of the input data based on its features. There are several classification algorithms that are commonly used in supervised learning. Some of the most widely used classification algorithms are:

1. **Decision Tree:**

   An effective way to introduce the concept of decision trees is to explain that they emulate human cognition by using a series of questions and answers to classify or predict an outcome based on the input features. The more complex a situation comes, the deeper and wider the tree becomes as shown in figure 2.2.

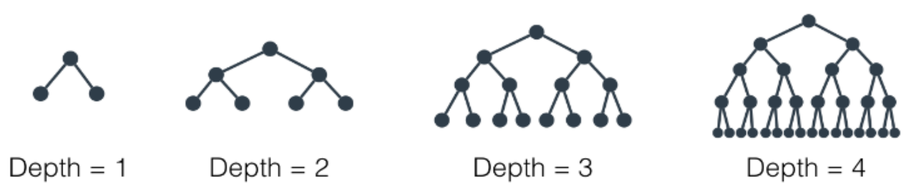


**Figure 2.2: Decision Tree** [9]

   A decision tree algorithm that people may encounter in their daily routine is exemplified by watching a documentary or reading an article on a particular topic. For instance, if someone watches a documentary on a certain historical era or reads about how neurons transmit information in the brain, YouTube or Google may use a decision tree algorithm to suggest related articles or videos that match their interests.

2. **Random Forest:**

   The random forest classifier is an ensemble learning method used for classification tasks. It derives its name from the fact that it consists of a large number of individual decision trees. These trees are constructed independently and their outputs are combined to make the final prediction as shown in Figure 2.3. Ensemble learning refers to the technique of combining multiple machine learning models to improve their performance and accuracy.


**Figure 2.3: Visualization of a Random Forest Model Making a Prediction on a coronal slice [10]**

Random Forest can be used for fraud detection in various industries such as finance, insurance, and e-commerce. It can analyze large datasets to identify patterns and anomalies that may indicate fraudulent activity. For example, if a particular customer's transaction history deviates significantly from the norm or if certain transactions occur at unusual times or locations, it may trigger a fraud alert.

3. **Naive Bayes:**

   Naive Bayes is a supervised machine learning algorithm that can be trained to classify data into multi-class orders. In the heart of the Naive Bayes algorithm is the probabilistic model that computes the tentative chances of the input features and assigns the probability distributions to each of the possible classes. This algorithm has great benefits similar to being easy to apply and fast to train [11].

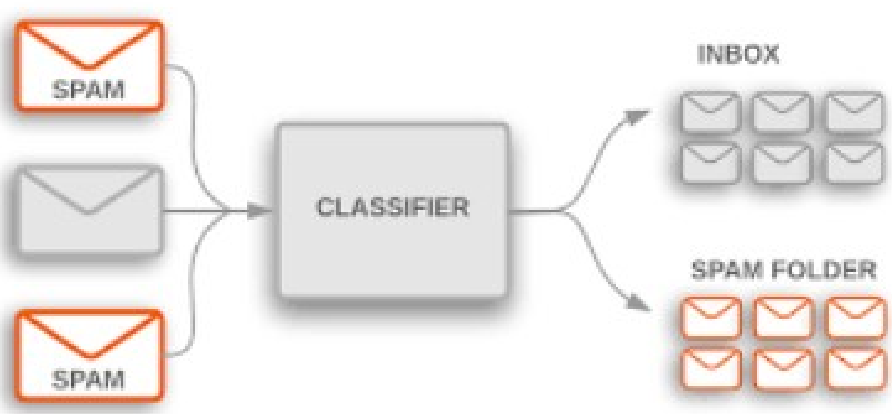


**Figure 2.4: Detecting spam emails using Naive Bayes classification algorithm**

The Naive Bayes algorithm is utilized in various applications such as spam filtering, where it calculates the probability of an email being spam based on certain words or phrases, like "easy" and "money", as illustrated in Figure 2.4. Additionally, [12] demonstrates a probability-based rainfall prediction, which takes into account factors like humidity, temperature, wind direction, and speed to predict whether it will rain in a specific area using a Naive Bayes model.

4. **Support Vector Machines (SVM):**

   As shown in Figure 2.5, SVM splits data with a line and it adds margins into the equation as a tool to further improve the accuracy of the training model. It finds the best hyperplane that separates the data into different classes by maximizing the margin between the hyperplane and the closest data points [9].

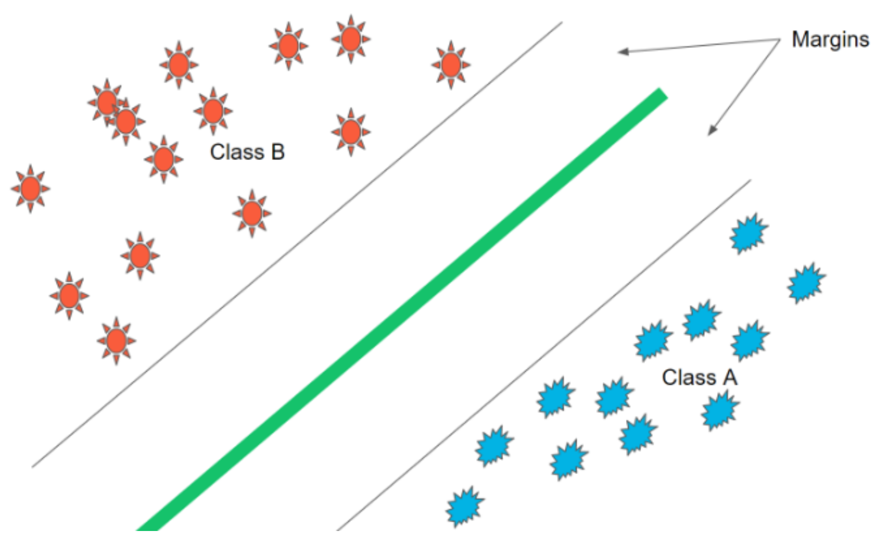


**Figure 2.5: Support Vector Machines Classification Algorithms**

One common use of SVM is in text classification, such as sentiment analysis. SVM can be used to classify whether a given text (such as a review or tweet) expresses a positive or negative sentiment. In this case, the SVM is trained on a labeled dataset of text samples, where the output labels indicate the sentiment expressed in the text.

5. **K-Nearest Neighbors (KNN):**

   This algorithm is used for classification and regression tasks [13]. It is a non-parametric algorithm, which means it does not make any assumptions about the distribution of the data. KNN works by finding the k nearest data points to a new input data point and then classifying the new data point based on the class of those k nearest neighbors. In other words, KNN uses the majority class among the k nearest neighbors to predict the class of the new data point. The value of k is a hyperparameter that can be tuned to optimize the performance of the algorithm. KNN is a simple and intuitive algorithm, but it can become computationally expensive for large datasets, and the choice of k can have a significant impact on the accuracy of the predictions. KNN has many applications, including image recognition, text classification, and recommendation systems.

   An example of the KNN classifier is image classification, where a dataset of images featuring creatures resembling cats and dogs can be used to determine whether a given creature in an image is a cat or a dog. The KNN algorithm operates based on a similarity measure, making it suitable for this type of task as shown in figure 2.6.


**Figure 2.6: KNN Classifier [14]**

#### 2.1.1.2 Regression

The algorithm learns to predict a numerical value or a continuous output variable based on the input features. There are several regression algorithms that are commonly used in machine learning. Some of the most widely used regression algorithms are:

1. **Linear Regression:**

   Linear regression is used for predicting a continuous outcome variable based on one or more input variables that are continuous or categorical [15]. It assumes that there is a linear relationship between the input variables and the outcome variable. In other words, linear regression aims to find the best-fitting straight line that describes the linear relationship between the dependent variable and the independent variable(s). As shown in Figure 2.7, the objective of linear regression is to find the line of best fit that minimizes the sum of the squared errors between the predicted values and the actual values.

   One of the most common examples of linear regression is predicting house prices based on various features such as the number of bedrooms, square footage,

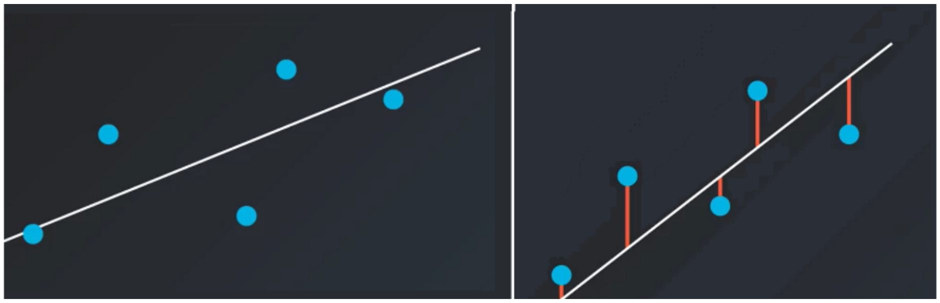

**Figure 2.7: Linear Regression**

location, etc. The idea is to fit a line (or hyperplane in higher dimensions) through the data points that best represents the relationship between the input features and the target variable (house price in this case). Once the line is fitted, it can be used to predict the house price for new houses based on their input features.

2. **Logistic Regression:**

   Logistic regression is used to predict a binary outcome variable based on one or more input variables that can be continuous or categorical [16]. It models the probability of the binary outcome variable as a function of the input variables, using a logistic function. The objective of logistic regression is to find the values of the coefficients of the logistic function that maximize the likelihood of the observed data. It learns a linear relationship from the given dataset and introduces a non-linearity in the form of the logistic sigmoid function to model the probability of the binary outcome as shown in figure 2.8.

   One example of logistic regression is predicting whether a customer will purchase a product or not based on their demographic and behavioral data such

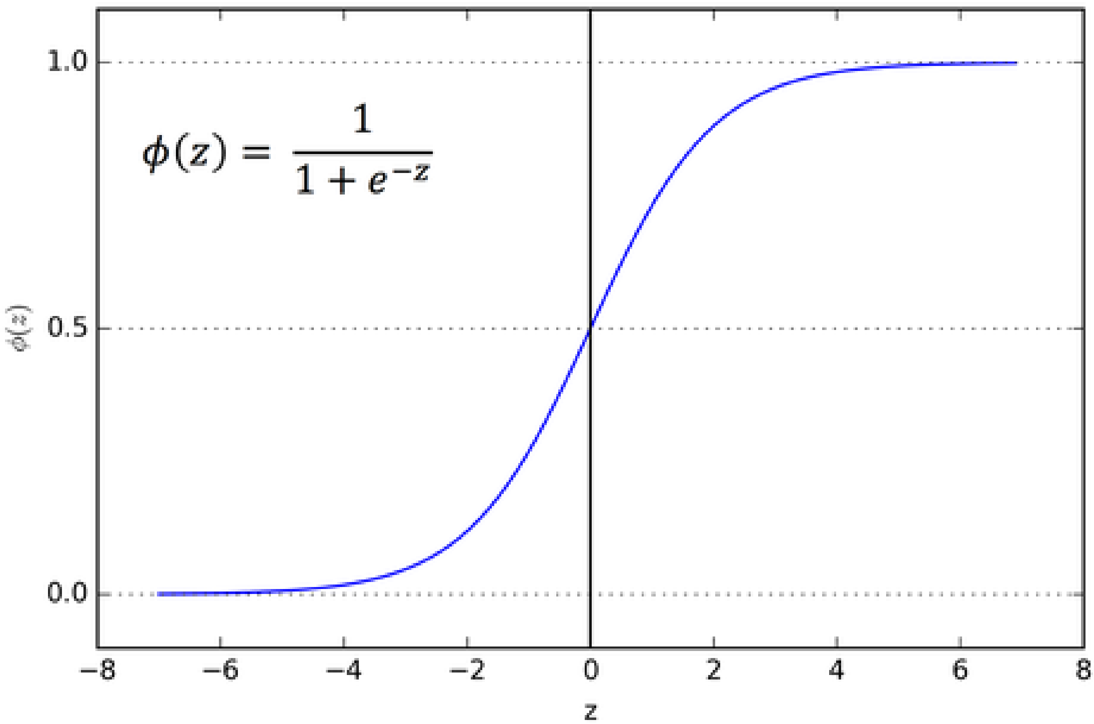


**Figure 2.8: Logistic sigmoid function**

as age, gender, income, past purchase history, etc. Another example is predicting whether a patient has a particular disease based on their medical history, symptoms, and other factors.

### 2.1.2 Unsupervised Machine Learning

Unsupervised learning is a type of machine learning in which the algorithms are trained on unlabeled data, meaning that the input data has no known output or target variable. The goal of unsupervised learning is to find patterns and structures in the data and to group similar instances together.

Unsupervised learning is all about understanding how to effectively group data, if the following took place:

1. Do not have a label to predict. An example of this is analyzing brain scans to identify areas that may indicate potential health concerns. Since there are no labels on the images, it can be difficult to determine which areas may be problematic. However, an algorithm can group areas based on their similarity or dissimilarity, which allows us to identify potential issues.

2. Are not trying to predict a label, but rather group data together. An example of this is when you have a large number of features, and you want to condense it down to a smaller set of features to be used.

The two main types of unsupervised learning are clustering and dimensionality reduction:

1. **Clustering:**

   is a process of grouping similar data points together into clusters based on their intrinsic properties or similarities. The goal is to identify patterns and structures in the data without prior knowledge of the class labels or output variables. Examples of clustering algorithms include k-means, hierarchical clustering, and density-based clustering.

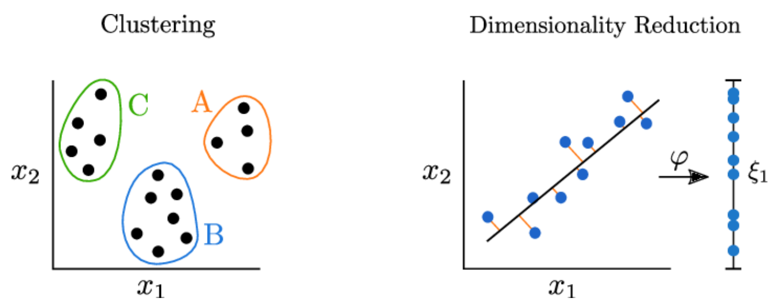


**Figure 2.9: Clustering vs. Dimensionality Reduction**

2. **Dimensionality Reduction:** is a process of reducing the number of input features while preserving the most important information or structure of the data. The goal is to simplify the data representation and improve the computational efficiency of subsequent analysis. Examples of dimensionality reduction techniques include Principal Component Analysis (PCA), Random Projection, and Independent component analysis (ICA)

   2.1. **Random Projection:**

   As shown in Figure 2.9, Random Projection is a technique used for reducing the number of dimensions in a dataset while maintaining its structural integrity. The approach involves projecting the original data onto a lower-dimensional space that is selected randomly. This projection can be viewed as a linear transformation that seeks to preserve the distance between the data points to the greatest extent possible. Compared to PCA, Random Projection is a more computationally efficient method for reducing the number of dimensions in large datasets. The dimension of the transformed data is determined by an error term, epsilon, which governs the amount of distance or information from the original dataset that is preserved.

## 2.2 Introduction of Hardware Trojan

A hardware Trojan refers to the malicious modification of a chip or integrated circuit during the manufacturing process, which can compromise the security and functionality of the hardware. Hardware Trojans can be introduced by insiders, such as employees or contractors, or by attackers who gain access to the manufacturing facilities. Hardware Trojans can take many forms, including modifications to the logic or functionality of the chip, insertion of additional circuitry, or changes to the power or timing characteristics of the device. Once a Trojan is implanted, it can be activated remotely to perform a variety of malicious activities, such as stealing data, altering system behavior, or even rendering the device inoperable.

Hardware Trojans are a significant concern for the security of critical systems such as military and aerospace applications, financial systems, and other systems where the integrity of the hardware is crucial. The detection and prevention of hardware Trojans is an active area of research in the fields of hardware security and trusted manufacturing.

### 2.2.1 Characterization of Hardware Trojans

In Figure 2.10 illustrate multiple methods to classify Hardware Trojans (HT), which are based on their characteristics and behavior, including their Physical Characteristics, Activation Mechanism, and Action Phase (Effect).

1. **Physical Characteristics**

   Trojans can be classified based on their physical characteristics, which can either be functional or parametric in nature.

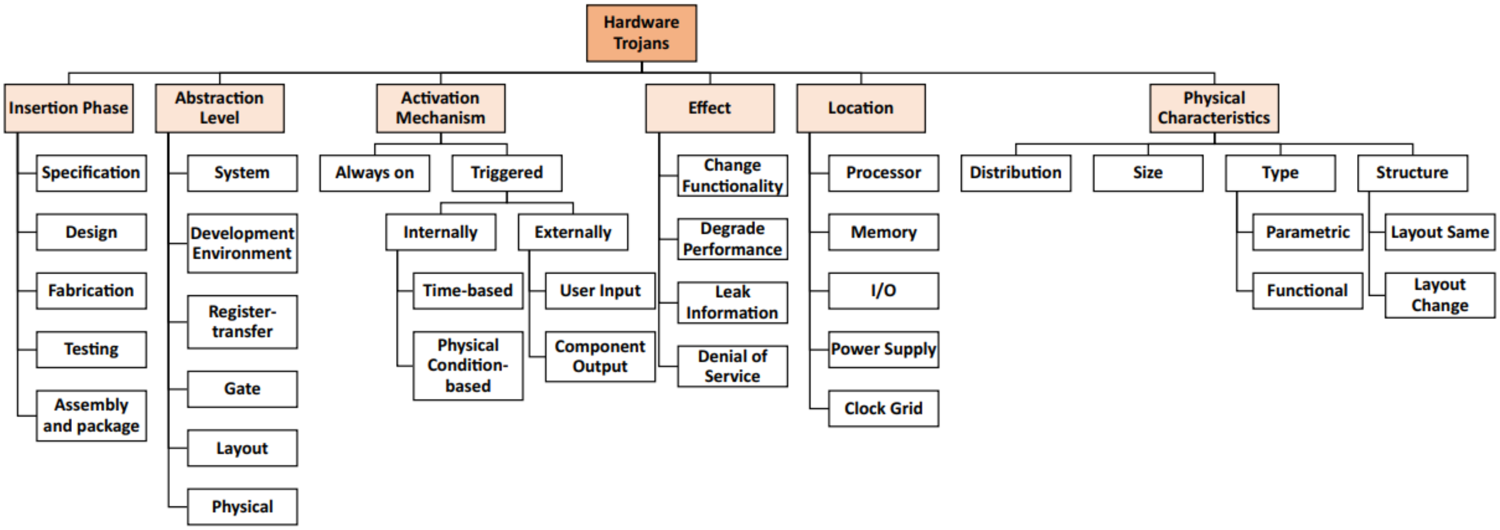


**Figure 2.10: Comprehensive HT taxonomy [7]**

(a) A functional Trojan is a type of hardware Trojan that involves the addition or deletion of gates or flip-flops from the original design, allowing an attacker to gain unauthorized access to the device or modify its behavior in some way. An example of a functional Trojan is the insertion of malicious code into a microprocessor.

(b) Parametric Trojan - modify the original circuitry. For example: diluting flip-flops, or subjecting the chip to radiation to reduce the reliability of the chip.

2. **Activation Mechanism**

Hardware Trojans (HT) are initiated by an event or condition called an "activation mechanism", which can be classified based on factors like a specific input sequence, operating condition, or time. There are various methods to trigger HTs, including:

(a) Internally activated where the malicious circuitry in the chip awakes the Trojan after a specific period of time, as illustrated in Figure 1.2.

(b) Externally activated where the malicious logic placed inside the chip uses an antenna or other sensors to allow the adversary to access the design externally. One example of this type of Hardware Trojan is when it's hidden in the control system of a cruising missile, enabling the attacker to remotely access and manipulate the system [4].

3. **Action Phase (Effect)**

Hardware Trojans can be categorized based on their effects, which refers to the behavior displayed by the Trojan once it has been activated. This behavior may involve data theft or modification, disruption of the system's normal operation, denial of service, or unauthorized hardware access.

## 2.3 Survey of datasets and features used in Hardware Trojan detection

Hardware Trojan detection using machine learning requires the availability of datasets containing good circuits, which serve as a reference model, and circuits with Trojans, which serve as the target for detection. In addition, the selection of appropriate features that capture the circuit's behavior is crucial for the success of the machine learning algorithm. In this section, I will provide a survey of some of the commonly used datasets and features in hardware Trojan detection.

My dataset was built on the Hardware Trojan Benchmarks. The benchmarks used in this thesis are from trust benchmark [7], which is a benchmark circuit (composed of generic circuits at the RTL, gate, or layout level) that intentionally incorporates Trojans at difficult-to-detect, significant, and/or opportunistic locations (e.g., uncommon nodes, layout white-space, etc.). The Trojans are inserted at different stages of the design process and vary in their types and functionalities, including stealthy Trojans, combinational Trojans, and sequential Trojans. The dataset also provides several sets of features, such as Number of cells, number of buf/inv, Total cell area, leakage power, power consumption measurements. It contains 907 digital circuits with 21 Trojan-free circuits and 886 Trojan-infected circuits.

## 2.4 Literature Review

HT detection has been and still is a back-and-forth tug of war. Whenever a new Hardware Trojan detection method capable of detecting current HTs is proposed, a new trojan emerges to bypass the current detection method.

One of the earliest shield mechanisms against Hardware Trojan approaches is proposed by Hicks etal [17] called Unused Circuit Identification (UCI) mechanism that can identify the suspicious circuitry during design verification. A year later, a new design emerged by Sturton etal [18] called Stealthy and Malicious Circuit (SMC) to bypass the UCI method by hiding HT in nearly-unused logic. Rajendran et al [19] proposed a detection method that detects information leaking Trojans and produces the trigger condition for the Trojan. His technique was able to detect a leak in the cryptographic key for AES-600 but failed in detecting a leak in the cryptographic key for AES-T1200.

Even though numerous methods for detecting Hardware Trojans have been proposed in the literature, a reliable and efficient approach to identifying emerging Hardware Trojans is still needed. In [20], the discussion centered around the importance of supervised and unsupervised learning in IoT, as well as the limitations of various machine learning techniques due to overfitting of the machine learning models in ensuring IoT security.

In [21], a machine learning technique was suggested for identifying Hardware Trojans by examining power consumption. Furthermore, a specialized many-core platform was constructed, which utilized a supervised machine learning algorithm powered by SVM to recognize communication attacks activated by HTs. This approach attained an accuracy rate of 94% [22]. Nonetheless, these approaches concentrated solely on detecting HTs during run-time and didn't involve extracting Trojan features from the design netlist, which would be more effective since the gate-level netlist includes a comprehensive list of gate and IP connections with functional and timing behaviors of the design.

In [23], information entropy-based clustering was employed, with the feature threshold set for Trojan detection. The DBSCAN model was used in [24] without setting the feature threshold value to detect Hardware Trojans. Another clustering method is proposed in [25] based on fuzzy logic for cryptographic applications. Authors of [23],[24], and[25] applied their techniques on a few types of HT circuits, also they encountered low accuracy due to the linearity between features caused by scaling the dataset in machine learning.

In[26], a total of fifty-one trojan features were extracted for HT detection. The best 11 trojan features were manually selected to be used as inputs for the Random forest classifier algorithm, resulting in an accuracy of 74.6% on only 12 benchmarks. Hasegawa et al. [27] focused on detecting hardware Trojans in pre-silicon using a machine-learning-based model. They extracted net features from the design post-synthesis and utilized a supervised machine learning algorithm based on Support Vector Machine (SVM) to distinguish between Trojan-free and infected designs. However, due to the linearity between features caused by scaling the dataset in machine learning, the effectiveness of this approach was limited resulting in an 85.28% for TPR.

A Neural Network model was employed in [28] utilizing eleven Trojan netlist features to detect hardware Trojans. However, the approach failed to be effective because of the linearity between the features caused by machine learning, resulting in a TNR accuracy of 59.5%. On the other hand, the RG-Secure framework, introduced in [29], implemented a lightweight gradient lifting algorithm to detect hardware Trojans concurrently at the register-transfer level and gate-level netlist. Although the accuracy of this approach was found to be high, it was only effective on a limited number of hardware designs, with one instance of hardware Trojan exhibiting a detection rate below 60%.

Multi-layer back propagation neural networks and one-class SVM were proposed in [30] to detect hardware Trojans (HTs) utilized for information leakage and identify their precise location in the design. This approach achieved a True Positive Rate (TPR) of 85.05% and a True Negative Rate (TNR) of 73.91%. Additionally, an unsupervised machine learning algorithm combining Principal Component Analysis (PCA) and Local Outlier Factor (LOF) algorithm for Trojan Detection at the gate-level-netlist, named PL-HTD, was introduced in [31]. Due to overfitting the training dataset, this approach showed an average true positive rate of 42.42%.

# CHAPTER 3: FEATURE EXTRACTION AND ML ALGORITHM SELECTION

## 3.1 preliminaries

This chapter covers the technique used to extract features from the hardware design of a chip, as well as the selection of a machine learning algorithm for detecting HT embedded in the design.

## 3.2 Introduction

Hardware description languages (HDLs) are specialized programming languages used in digital circuit design to describe the behavior and structure of digital circuits at various levels of abstraction, from high-level system design to low-level gate-level netlists. Verilog and VHDL are the most commonly used HDLs by industry, and they are used to model digital systems, simulate, and synthesize them into real hardware. These languages allow designers to describe complex digital circuits with ease, providing a way to verify their functionality before manufacturing. HDLs are an essential part of the digital design process and are used extensively in the semiconductor industry to design and verify the functionality of digital circuits.

## 3.3 Hardware Trojan Detection Flow Diagram

Figure 3.1 shows the proposed HT detection scheme which consists of the following steps:

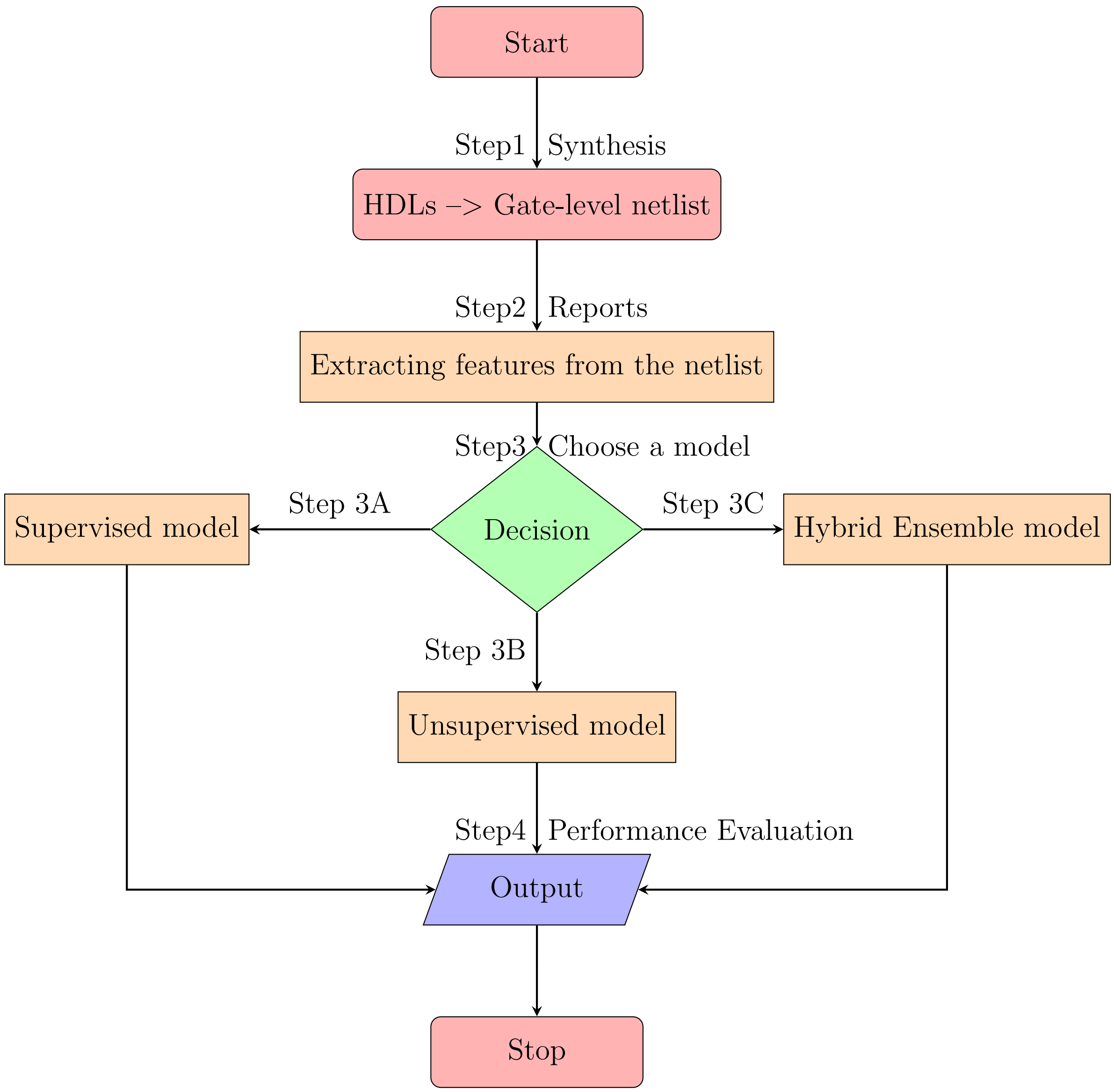


**Figure 3.1: Hardware Trojan detection scheme**

1. **Step 1:** Synthesis

   The process is initiated by writing a Tcl script that tests the behavioral Verilog to ensure that the digital hardware design meets the specified constraints for timing, power, and area. Violations of these constraints may occur if the design does not meet the maximum time delay, power consumption, or area limits. The Tcl script used to accomplish this task sets the clock period to 20000 ps to achieve an operating frequency of 50 MHz and defines the use of a 45nm technology package.

2. **Step 2:** Feature Extraction

   The Cadence genus tool was employed to produce several reports that detail the design's timing, power, and area, as depicted in Figures D.1 , D.2, D.4 and D.3 included in Apendix D. Each feature includes subsets of components utilized to compute the constituent parts of the feature, as illustrated in Table 3.1. A total of Thirty-One netlist features are extracted to construct a database that is then used to train machine learning models for discerning whether the design is contaminated with a trojan or not.

3. **Step 3:** Decision (Choosing a Machine Learning Model)

   The focus of this stage is to establish the methodology for training on netlist features, which comprises three distinct procedures. The best-performing method is retained for future testing. The initial procedure entails applying a supervised machine learning approach, the second entails utilizing an unsupervised machine learning approach, and the third involves implementing a hybrid ensemble model.

**Table 3.1: Extracted Features for Trojan Detection**

| Features extracted for detection | |
|---|---|
| **Area** | **Power consumption** |
| Number of ports | Cell Internal Power |
| Number of nets, | Net Switching Power |
| Number of cells, | Total Dynamic Power |
| Number of combinational cells | Cell Leakage Power |
| Number of sequential cells | Register Internal Power |
| Number of buf/inv | Register Switching Power |
| Number of references | Register Total Power |
| Combinational area | Total Switching Power |
| Buf/Inv area | Sequential Internal Power |
| sequential area | Total Power |
| Total cell area | Sequential Leakage Power |
| | Combinational Internal Power |
| | Combinational Switching Power |
| | Combinational Leakage Power |
| | Combinational Total Power |
| | Register Leakage Power |
| | Total Internal Power |
| | Total Leakage Power |
| | Sequential Switching Power |
| | Sequential Total Power |

**Step 3A: Supervised machine learning model**

(a) **Step 3A.1:** Dropping step

In this stage, the correlation coefficient is utilized to evaluate the degree of linear relationship between features. Features that exhibit high correlations are considered to be linearly dependent, with a perfect positive correlation represented by a value of 1 and a perfect negative correlation represented by a value of -1. Features that do not display a linear relationship are regarded as independent, with a value of 0 denoting no linear correlation. To prevent overfitting of the machine learning algorithm on the training dataset, one of the two features with high correlation is removed. Heat maps are employed to demonstrate the interdependence between features, as depicted in figures D.5 and D.6.

(b) **Step 3A.2:** Data Shuffling

Shuffling the datasets is crucial prior to training in order to prevent the model from learning a specific pattern. This helps to decrease variance and enables the model to perform well on unfamiliar data.

(c) **Step 3A.3:** MinMaxScaler

The MinMaxScaler class is utilized to scale the datasets, where each feature is scaled and shifted independently to ensure that it falls within the range of (0,1) in the training set. The use of MinMaxScaler is advisable if a normal distribution is desired, and the effect of outliers is minimized.

(d) **Step 3A.4:** Random Forest Classifier

The random forest classifier is a meta-estimator that involves fitting multiple decision tree classifiers on different sub-samples of the dataset. This technique employs averaging to enhance the predictive accuracy and mitigate overfitting, making it suitable for classifying whether a feature is infected with a Trojan or not.

**Step 3B Unsupervised machine learning model**

(a) **Step 3B.1:** Removing Labels

Removing labels is crucial in the approach of using an unsupervised machine learning model, as it aims to reduce bias in the model. The model is designed to identify patterns or clusters in the data with minimal human involvement.

(b) **Step 3B.2:** Shuffling Data

Similar to the supervised learning model, the datasets must be randomized in order to ensure that the model can effectively generalize to unfamiliar data.

(c) **Step 3B.3:** Random Projection

At this stage, utilizing the Sparse Random Projection method reduces the dimensionality of the datasets in Euclidean space, which not only ensures consistent embedding quality but also enhances the speed of computation for the projected data. The quality of the dimensionality reduction is controlled by epsilon in the Sparse Random Projection. Epsilon determines the level of distortion allowed in the projection process, meaning it specifies the acceptable error when approximating the high-dimensional data in the lower-dimensional space.

(d) **Step 3B.4:** Random Forest Classifier

By using Random Forest Classifier after applying Sparse Random Projection in the Hardware Trojan detection process, the impact of irrelevant or redundant features in the data can be minimized, which reduces the risk of overfitting and improves the generalization performance of the algorithm.

- **Step 3C: Hybrid Ensemble Model**

  The Hybrid Ensemble model is a type of machine learning model that integrates multiple models from diverse families to generate predictions. In the present study, the following models were employed: Logistic Regression, Decision Tree, Support Vector Machine, K-Nearest Neighbor, and Naive Bayes, as illustrated in Figure D.7.

  Each model in the hybrid ensemble is configured with distinct parameters. For instance, the Decision Tree Classifier has varying maximum depth settings of 2, 3, 4, and 5. The Logistic Regression model applies L2 regularization to penalize the sum of squares of the weights. In the case of the Support Vector Classifier (SVC), kernel parameters include linear, poly, and Radial Basis Function (RBF). Likewise, K-Nearest Neighbor and Naive Bayes models also have specific parameter settings.

  Then the voting classifier employs a hard voting ensemble approach where it aggregates the votes for discrete class labels from other models and makes predictions based on the class with the highest number of votes.

4. **Step 4:** Results

   The last stage of the hardware trojan detection process involves evaluating the performance of each model to ensure accuracy, which is achieved using the confusion matrix. This metric is commonly used in Machine Learning classification problems that have multiple output classes. However, since this is a binary classification problem where the output is either "Free Trojan" (0) or

"Trojan-infected" (1), the confusion matrix comprises four possible combinations of predicted and actual values, as shown in Figure 3.2.

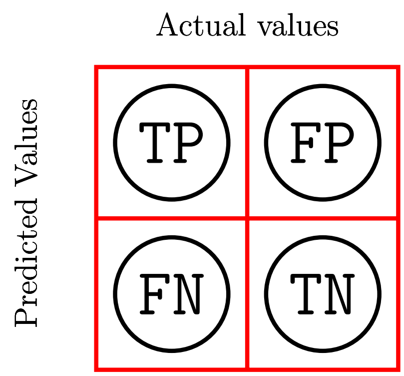


**Figure 3.2: Confusion Matrix**

Specifically, TP indicates that the model correctly predicts a Trojan-infected chip, TN indicates that the model correctly predicts a Trojan-free chip, FP indicates that the model wrongly predicts a Trojan-infected chip, and FN indicates that the model wrongly predicts a Trojan-free chip.

For further explanation on how the confusion matrix is used to evaluate the model's accuracy, precision, recall, and F1 score (F-measure), which are all important performance metrics for evaluating classification models. Here is the formula of each metric and explanation:

(a) **Accuracy**

is simply a measurement of how often the model makes a correct prediction. It is calculated by dividing the number of correct predictions by the total number of predictions made. **Accuracy** = (TP + TN) / (TP + TN + FP + FN)

(b) **Precision**

is simply a measure of how many times the model achieved correct prediction out of all the total positive predicted.

**Precision** = TP / (TP + FP)

(c) **Recall**

measures the proportion of actual positive instances that are correctly classified as positive by the model. A high recall indicates that the model is able to identify most of the positive instances correctly. A low recall indicates that the model is missing a significant number of actual positive instances. Recall is an important metric, especially in situations where identifying positive instances is critical, such as hardware trojan detection.

**Recall** = TP / (TP + FN)

(d) **F-measure (F1 Score)**

is a measure of a classification model's performance that takes both Precision and Recall into account. It is the harmonic mean of Precision and Recall and is calculated as:

**F1 Score** = 2 * (Precision * Recall) / (Precision + Recall)

The F1 score provides a balance between Precision and Recall, making it a useful metric for evaluating models in situations where both high Precision and high Recall are important. For example, in a medical diagnosis scenario, both high Precision (correctly identifying those with the condition) and high Recall (correctly identifying all those with the condition) are crucial for an accurate diagnosis.

# CHAPTER 4:
# PERFORMANCE EVALUATION AND RESULTS

## 4.1 Preliminaries

In this chapter, the results of detecting hardware Trojans using the machine learning classification models are presented. Furthermore, the number of features integrated into the model after eliminating linearity among them is included, along with accuracy and precision measurements obtained through the confusion matrix.

## 4.2 Results

**Table 4.1: RESULTS OF MACHINE-LEARNING-BASED CLASSIFICATION**

| Approach | N-Features | TN | FP | FN | TP | TPR | TNR | precision | F-measure | Recall |
|---|---|---|---|---|---|---|---|---|---|---|
| Supervised | 9 | 280 | 2 | 5 | 622 | 99.2% | 99.2% | 99.6% | 99.3% | 99.2% |
| Unsupervised | 3 | 282 | 1 | 3 | 623 | 99.5% | 99.6% | 99.8% | 99.6% | 99.5% |
| Hybrid Ensemble | 9 | 27 | 14 | 239 | 629 | 72.47% | 65.85% | 97.82% | 83.26% | 72.47% |

The performance evaluation results for the classification models are presented in Table 4.1. When the supervised learning model was applied using only nine Trojan features, it achieved a 99.2% true positive rate (TPR) and true negative rate (TNR) on all netlists, successfully classifying them as either Trojan-Infected or Trojan-Free. On the other hand, the unsupervised learning model approach identified a 99.5%

TPR and 99.6% TNR using only three Trojan features. The hybrid ensemble model yielded a TPR of 70.07% and a TNR of 0%.

## 4.3 Comparison of the proposed approach with existing methods

**Table 4.2: COMPARISON TO AN EXISTING METHODS**

| Approach/Paper | | Precision | F-measure | TPR | TNR |
|---|---|---|---|---|---|
| Supervised | [27] | 2.8% | 5.2% | 85.28% | 52.67% |
| Supervised | [28] | - | - | 85% | 70% |
| Supervised | [26] | 92.2% | 74.6% | 68.32% | 99.7% |
| Unsupervised | [32] | - | 93.9% | - | - |
| Supervised | Ours | 99.1% | 98.8% | 99.2% | 98.8% |
| Unsupervised | Ours | 99.5% | 99.4% | 99.5% | 99.6% |
| Hybrid Ensemble | Ours | 97.82% | 83.26% | 72.47% | 65.85% |

Table 4.2 compares the proposed hardware Trojan detection models with other existing approaches in terms of precision, F-measure, TPR, and TNR. The proposed approaches outperformed the other methods, suggesting that removing linearity among the features improved the model's ability to achieve higher accuracy in detecting hardware Trojans on the test dataset.

# CHAPTER 5:

# CONCLUSION AND FUTURE WORK

## 5.1 Conclusion

In this thesis, two machine learning algorithms are used to detect Hardware Trojans from the extracted features of hardware designs before and after embedding a HT. These features were extracted using the Genus tool in Cadence, which provides descriptions of the hardware design such as power, area, gate, and timing analysis, as explained in chapter 3, step 2 of feature extraction.

To detect Hardware Trojans, the proposed method incorporates three machine learning models: a supervised model, an unsupervised model, and a hybrid ensemble model. The supervised model employs Thirty-Three Trojan features and analyzes the linearity between these features using a correlation matrix. Conversely, the unsupervised model utilizes Random Projection to randomly select fewer features, thereby enhancing accuracy when a Random Forest Classifier is employed. Additionally, a Hybrid Ensemble model is introduced that integrates multiple models from distinct families, such as Logistic Regression, Decision Tree, Support Vector Machine (SVM), K-Nearest Neighbor (KNN), and Naive Bayes.

The supervised learning model achieved a maximum TPR of 99.2% and TNR of 98.8%, whereas the unsupervised learning model outperformed it with a TPR of

99.5% and TNR of 99.6% across all benchmarks. On the other hand, the Hybrid Ensemble model obtained a lower TPR of 71.73%.

## 5.2 Future Work

In this section, we explore potential directions for future research in Hybrid Ensemble models for hardware trojan detection. One area of focus is the investigation of various models that can be combined to enhance accuracy, including traditional machine learning models like decision trees and support vector machines, as well as deep learning models like convolutional neural networks and recurrent neural networks.

Another research area is the development of more resilient ensemble methods that can manage variations in hardware design. This could entail using techniques such as adversarial training to bolster model resistance against tampering and attacks.

Lastly, there is a need for scalable ensemble methods for hardware trojan detection, particularly as hardware designs become more intricate. This could involve the use of distributed ensemble methods, such as federated learning, that can process vast amounts of data in parallel while preserving privacy.

In conclusion, future research holds immense potential for developing more robust and accurate Hybrid Ensemble methods for hardware trojan detection, which will help safeguard the security and integrity of hardware systems.

# REFERENCES


[1] Rachel Selina Rajarathnam, Yibo Lin, Yier Jin, and David Z. Pan. Regds: A reverse engineering framework from gdsii to gate-level netlist. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 154–163, 2020. doi: 10.1109/HOST45689.2020.9300272.

[2] Sina Faezi, Rozhin Yasaei, Anomadarshi Barua, and Mohammad Abdullah Al Faruque. Brain-inspired golden chip free hardware trojan detection. *IEEE Transactions on Information Forensics and Security*, 16:2697–2708, 2021. doi: 10.1109/TIFS.2021.3062989.

[3] Sally Adee. The hunt for the kill switch. *iEEE SpEctrum*, 45(5):34–39, 2008.

[4] A Rawnsley. Fishy chips: Spies want to hack-proof circuits. *Wired*, 2011.

[5] Sergei Skorobogatov and Christopher Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 23–40. Springer, 2012.

[6] The big hack how china used a tiny chip to infiltrate americas. URL `https://www.bloomberg.com/news/features/2018-10-04/the-bighack-how-china-used-a-tiny-chip-to-infiltrate-america-s-topcompanies?leadSource=uverify/wall`.

[7] Bicky Shakya, Tony He, Hassan Salmani, Domenic Forte, Swarup Bhunia, and Mark Tehranipoor. Benchmarking of hardware trojans and maliciously affected circuits. *Journal of Hardware and Systems Security*, 1, 03 2017. doi: 10.1007/s41635-017-0001-6.

[8] Regression vs. classification in machine learning , availiable: https://www.javatpoint.com/regression-vs-classification-in-machine-learning.

[9] Madan Somvanshi, Pranjali Chavan, Shital Tambade, and S. V. Shinde. A review of machine learning techniques using decision tree and support vector machine. In *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, pages 1–7, 2016. doi: 10.1109/ICCUBEA.2016.7860040.

[10] Random forest algorithm for the classification of neuroimaging data in alzheimer's disease: A systematic review. URL `https://www.frontiersin.org/articles/10.3389/fnagi.2017.00329/full`.

[11] K Ming Leung et al. Naive bayesian classifier. *Polytechnic University Department of Computer Science/Finance and Risk Engineering*, 2007:123–156, 2007.

[12] James NK Liu, Bavy NL Li, and Tharam S Dillon. An improved naive bayesian classifier technique coupled with a novel input solution method [rainfall prediction]. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 31(2):249–256, 2001.

[13] Padraig Cunningham and Sarah Jane Delany. k-nearest neighbour classifiers: 2nd edition (with python examples). *CoRR*, abs/2004.04523, 2020. URL `https://arxiv.org/abs/2004.04523`.

[14] K-nearest neighbor. URL `https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4`.

[15] David J Olive and David J Olive. *Multiple linear regression*. Springer, 2017.

[16] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

[17] Matthew Hicks, Murph Finnicum, Samuel T King, Milo MK Martin, and Jonathan M Smith. Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. pages 159–172, 2010.

[18] Cynthia Sturton, Matthew Hicks, David Wagner, and Samuel T King. Defeating uci: Building stealthy and malicious hardware. pages 64–77, 2011.

[19] Jeyavijayan Rajendran, Vivekananda Vedula, and Ramesh Karri. Detecting malicious modifications of data in third-party intellectual property cores. pages 1–6, 2015.

[20] Kushal Rashmikant Dalal. Analysing the role of supervised and unsupervised machine learning in iot. In *2020 international conference on electronics and sustainable communication systems (ICESC)*, pages 75–79. IEEE, 2020.

[21] Takato Iwase, Yusuke Nozaki, Masaya Yoshikawa, and Takeshi Kumaki. Detection technique for hardware trojans using machine learning in frequency domain. pages 185–186, 2015.

[22] Amey Kulkarni, Youngok Pino, and Tinoosh Mohsenin. Svm-based real-time hardware trojan detection for many-core platform. pages 362–367, 2016.

[23] Renjie Lu, Haihua Shen, Zhihua Feng, Huawei Li, Wei Zhao, and Xiaowei Li. Htdet: A clustering method using information entropy for hardware trojan detection. *Tsinghua Science and Technology*, 26(1):48–61, 2020.

[24] Pengyong Zhao and Qiang Liu. Density-based clustering method for hardware trojan detection based on gate-level structural features. pages 1–4, 2019.

[25] M Revanesh, V Sridhar, and John M Acken. Cb-alca: a cluster-based adaptive lightweight cryptographic algorithm for secure routing in wireless sensor networks. *International Journal of Information and Computer Security*, 11(6): 637–662, 2019.

[26] Kento Hasegawa, Masao Yanagisawa, and Nozomu Togawa. Trojan-feature extraction at gate-level netlists and its application to hardware-trojan detection using random forest classifier. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[27] Kento Hasegawa, Masaru Oya, Masao Yanagisawa, and Nozomu Togawa. Hardware trojans classification for gate-level netlists based on machine learning. In *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 203–206. IEEE, 2016.

[28] Kento Hasegawa, Masao Yanagisawa, and Nozomu Togawa. Hardware trojans classification for gate-level netlists using multi-layer neural networks. In *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 227–232. IEEE, 2017.

[29] Chen Dong, Guorong He, Ximeng Liu, Yang Yang, and Wenzhong Guo. A

multi-layer hardware trojan protection framework for iot chips. *IEEE Access*, 7: 23628–23639, 2019.

[30] Chen Dong, Fan Zhang, Ximeng Liu, Xing Huang, Wenzhong Guo, and Yang Yang. A locating method for multi-purposes hts based on the boundary network. *IEEE Access*, 7:110936–110950, 2019.

[31] Chen Dong, Yulin Liu, Jinghui Chen, Ximeng Liu, Wenzhong Guo, and Yuzhong Chen. An unsupervised detection approach for hardware trojans. *IEEE Access*, 8:158169–158183, 2020.

[32] Taifeng Hu, Liji Wu, Xiangmin Zhang, and Zhaopo Liao. Hardware trojan detection combines with machine learning: an isolation forest-based detection method. In *2020 IEEE 14th International Conference on Big Data Science and Engineering (BigDataSE)*, pages 96–103, 2020. doi: 10.1109/BigDataSE50710.2020.00021.

# APPENDIX A:

# HYBIRD ENSEMBLE PYTHON CODE

```
#Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
from sklearn import model_selection
from sklearn.metrics import confusion_matrix
from sklearn. preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.utils import shuffle
from sklearn.preprocessing import MinMaxScaler

#Reading the dataset
X=prepare_data("Benchmark_Feature_Extraction.xlsx")
display.display(X.columns)

X,y=preprocess_data (X)



# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)



#Defining the machine learning models
model1 = LogisticRegression()
model2 = DecisionTreeClassifier(max_depth = 2)
```

```
model3 = SVC()
model4 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p =
model5 = GaussianNB()

#Training the machine learning models
model1.fit(X_train, np.ravel(y_train,order='C'))
model2.fit(X_train, np.ravel(y_train,order='C'))
model3.fit(X_train, np.ravel(y_train,order='C'))
model4.fit(X_train, np.ravel(y_train,order='C'))
model5.fit(X_train, np.ravel(y_train,order='C'))

#Making the prediction
y_pred1 = model1.predict(X_test)
y_pred2 = model2.predict(X_test)
y_pred3 = model3.predict(X_test)
y_pred4 = model4.predict(X_test)
y_pred5 = model5.predict(X_test)

#Confusion matrix
cm_LogisticRegression = confusion_matrix(y_test, y_pred1)

sns.heatmap(cm_LogisticRegression, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value')
```

```
plt.savefig("./outputs/cm_LogisticRegression.png")
plt.show()

cm_DecisionTree = confusion_matrix(y_test, y_pred2)

sns.heatmap(cm_DecisionTree, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value')
plt.savefig("./outputs/cm_DecisionTree.png")
plt.show()


cm_SupportVectorClass = confusion_matrix(y_test, y_pred3)
sns.heatmap(cm_SupportVectorClass, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value')
plt.savefig("./outputs/cm_SupportVectorClass.png")
plt.show()


cm_KNN = confusion_matrix(y_test, y_pred4)
sns.heatmap(cm_KNN, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value')
```

```
plt.savefig("./outputs/cm_KNN.png")
plt.show()



cm_NaiveBayes = confusion_matrix(y_test, y_pred5)
sns.heatmap(cm_NaiveBayes, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value')
plt.savefig("./outputs/cm_NaiveBayes.png")
plt.show()


#10-fold cross-validation
kfold = model_selection.KFold(n_splits=10)
result1 = model_selection.cross_val_score(model1, X_train, np.ravel(y_tr
result2 = model_selection.cross_val_score(model2, X_train, np.ravel(y_tr
result3 = model_selection.cross_val_score(model3, X_train, np.ravel(y_tr
result4 = model_selection.cross_val_score(model4, X_train, np.ravel(y_tr
result5 = model_selection.cross_val_score(model5, X_train, np.ravel(y_tr



#Printing the accuracies achieved in cross-validation
```

```
print('Accuracy of Logistic Regression Model = ',result1.mean())
print('Accuracy of Decision Tree Model = ',result2.mean())
print('Accuracy of Support Vector Machine = ',result3.mean())
print('Accuracy of k-NN Model = ',result4.mean())
print('Accuracy of Naive Bayes Model = ',result5.mean())

#Defining Hybrid Ensemble Learning Model
# create the sub-models
estimators = []

#Defining 5 Logistic Regression Models
model11 = LogisticRegression(penalty = 'l2' )
estimators.append(('logistic1', model11))
model12 = LogisticRegression(penalty = 'l2' )
estimators.append(('logistic2', model12))
model13 = LogisticRegression(penalty = 'l2')
estimators.append(('logistic3', model13))
model14 = LogisticRegression(penalty = 'l2')
estimators.append(('logistic4', model14))
model15 = LogisticRegression(penalty = 'l2')
estimators.append(('logistic5', model15))

#Defining 5 Decision Tree Classifiers
model16 = DecisionTreeClassifier(max_depth = 3)
```

```
estimators.append(('cart1', model16))
model17 = DecisionTreeClassifier(max_depth = 4)
estimators.append(('cart2', model17))
model18 = DecisionTreeClassifier(max_depth = 5)
estimators.append(('cart3', model18))
model19 = DecisionTreeClassifier(max_depth = 2)
estimators.append(('cart4', model19))
model20 = DecisionTreeClassifier(max_depth = 3)
estimators.append(('cart5', model20))

#Defining 5 Support Vector Classifiers
model21 = SVC(kernel = 'linear')
estimators.append(('svm1', model21))
model22 = SVC(kernel = 'poly')
estimators.append(('svm2', model22))
model23 = SVC(kernel = 'rbf')
estimators.append(('svm3', model23))
model24 = SVC(kernel = 'rbf')
estimators.append(('svm4', model24))
model25 = SVC(kernel = 'linear')
estimators.append(('svm5', model25))

#Defining 5 K-NN classifiers
model26 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p
```

```
estimators.append(('knn1', model26))
model27 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p
estimators.append(('knn2', model27))
model28 = KNeighborsClassifier(n_neighbors = 6, metric = 'minkowski', p
estimators.append(('knn3', model28))
model29 = KNeighborsClassifier(n_neighbors = 4, metric = 'minkowski', p
estimators.append(('knn4', model29))
model30 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p
estimators.append(('knn5', model30))

#Defining 5 Naive Bayes classifiers
model31 = GaussianNB()
estimators.append(('nbs1', model31))
model32 = GaussianNB()
estimators.append(('nbs2', model32))
model33 = GaussianNB()
estimators.append(('nbs3', model33))
model34 = GaussianNB()
estimators.append(('nbs4', model34))
model35 = GaussianNB()
estimators.append(('nbs5', model35))

# Defining the ensemble model
ensemble = VotingClassifier(estimators)
```

```
ensemble.fit(X_train, y_train.ravel())
y_pred = ensemble.predict(X_test)

#Confisuin matrix
cm_HybridEnsembler = confusion_matrix(y_test, y_pred)
sns.heatmap(cm_HybridEnsembler, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value')
plt.savefig("./outputs/cm_HybridEnsembler.png")
plt.show()
#Cross-Validation
seed = 10

kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=Tru
results = model_selection.cross_val_score(ensemble, X_train, np.ravel(y_
print(results.mean())
```

# APPENDIX B:
# SUPERVISED AND UNSUPERVISED ML PYTHON CODE

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn

from mpl_toolkits.mplot3d import Axes3D
from sklearn import cluster
from sklearn.cluster import KMeans

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

from sklearn.utils import shuffle
import matplotlib.pyplot as plt
```

```
from sklearn.preprocessing import MinMaxScaler
from scipy.cluster.vq import kmeans
from scipy.spatial.distance import cdist,pdist
from matplotlib import cm
from IPython import display


from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import multilabel_confusion_matrix

#TO preforem random  projection to reduce the dataset dimension.
from sklearn.random_projection import SparseRandomProjection
from sklearn.random_projection import GaussianRandomProjection
def get_kmeans_score(data, center):
    '''
    returns the kmeans score regarding SSE for points to centers
    INPUT:
        data - the dataset you want to fit kmeans to
        center - the number of centers you want (the k value)
    OUTPUT:
        score - the SSE score for the kmeans model fit to the data
    '''
```

```
    #instantiate kmeans
    kmeans = KMeans(n_clusters=center)

    # Then fit the model to your data using the fit method
    model = kmeans.fit(data)

    # Obtain a score related to the model fit
    score = np.abs(model.score(data))

    return score

def fit_mods(data,n_max):
    scores = []
    centers = list(range(1,n_max))

    for center in centers:
        scores.append(get_kmeans_score(data, center))

    return centers, scores

def create_numerics(data):
    # Get nominal columns
    nominal_cols = data.select_dtypes(include='object').columns.tolist()
```

```
    # Turn nominal to numeric
    for nom in nominal_cols:
        enc = LabelEncoder()
        enc.fit(data[nom])
        data[nom] = enc.transform(data[nom])

    return data
def plot_data(data, labels):
    '''
    Plot data with colors associated with labels
    '''
    fig = plt.figure();
    ax = Axes3D(fig)
    ax.scatter(data[:, 0], data[:, 1], data[:, 2], c=labels, cmap='tab10


def Avg_score(data,n_max):
    scores=[]
    for centers in range(1,n_max):
        #Initaing muiltuiple models with different centers
        kmeans=KMeans(centers)
        #then fit the model with the data
        model=kmeans.fit(data)
        #Finally predcit the same data to show the point belongs to
```

```
            scores.append(abs(model.score(data)))
        centers=list(range(1,n_max))
        plt.plot(centers,scores)
        plt.title("scree plot")
        plt.xlabel("Centers")
        plt.ylabel("Average Distance from the centroid ")

        return  plt.show()
    def prepare_data(file_name):
        n=51
        #replace n with the number of columns you want to see completely
        pd.set_option('display.max_columns', n)
        #replace n with the number of rows you want to see completely
        pd.set_option('display.max_rows', n)
        data = pd.read_excel(file_name)

      # display.display(data["Circuit"])
        '''
        prepare the and inspect the data for unpersvised ans supervised:
            1.Unspervised Learning:
                1- "Cleanning stage" check for null ----<>Done
                2- Remove the label so that we can train the data
                3- check the data type for  object variables, if there is co
                4- Scale the data  --> K-means data prep using MinMaxScaler
```

```
        5- Convert back to a dataframe , and name the columns again
        6-

    '''
    ## drop the labels
    extracted_col=temp_netlist_data[['Label', 'Circuit']]
    ## new data set with average column to train on.
    temp_netlist_data_subset["Average"]=temp_netlist_data_subset.mean(ax
    temp_netlist_data_subset=temp_netlist_data_subset.join(extracted_col
    return temp_netlist_data_subset




def preprocess_data (data):
    ''' prepare the and inspect the data for unpersvised Machine Learnin
      1. Unspervised Learning:
        1- "Cleanning stage" check for null ---<>Done
        2- Remove the label so that we can train the data
        3- check the data type for  object variables, if there is co
        4-  balance the ratio between trojan free and infected of th
        5-
        6-
```

```
'''
#display.display(data.head())
data = data.dropna()
trojan_free = data.loc[data['Label']=="'Trojan_Free'"].reset_index()

# balance the ratio between trojan free and infected of the same cir
for i in range(len(trojan_free)):
    category_substring = trojan_free['Circuit'][i].replace("'",'')

    circuit_group = data[data['Circuit'].str.contains(category_subst

    df1 = circuit_group.iloc[0:1]
    if len(circuit_group) > 1:
        data = data.append([df1]*(len(circuit_group)-1), ignore_inde
data = create_numerics(data)
print(data)
data = shuffle(data, random_state=42)

# Create correlation matrix
corr_matrix = data.corr().abs()

#display.display(corr_matrix)
# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape),
```

```
                                          k=1).astype(np.bool))

    # Find index of feature columns with correlation greater than 0.95
    to_drop = [column for column in upper.columns if any(upper[column] >
   #print("to drop")
   # display.display(to_drop)
    # Drop features
    data = data.drop(data[to_drop], axis=1)
#   print(len(data))
    y_label = pd.DataFrame(data["Label"]).values
    x_data = data.drop(["Label","Circuit"], axis=1)

    corr_matrix_update=x_data.corr().abs()




    #display.display(x_data)

    scaler = MinMaxScaler(feature_range=(0, 1))
    x_data = scaler.fit_transform(x_data)

    x_train, x_test, y_train, y_test = train_test_split(x_data, y_label,
```

```
# plot the correlated features
 sns.heatmap(
       corr_matrix,
       vmin=-1, vmax=1, center=0,
       cmap=sns.diverging_palette(20, 220, n=200),
       square=False
 )
 plt.title("Features correlation")
 plt.savefig("Features correlation.png",dpi=300, bbox_inches='tight')

 plt.show()



 # plot the correlated features
 sns.heatmap(
       corr_matrix_update,
       vmin=-1, vmax=1, center=0,
       cmap=sns.diverging_palette(20, 220, n=200),
       square=False
 )
 plt.title("Features correlation after drop")
 plt.savefig("Features correlation after drop.png",dpi=300, bbox_inch
 plt.show()
 return x_data, y_label
```

```
def validation_tes(file_name2):



    data=prepare_data(file_name2)


    y_label = pd.DataFrame(data["Label"]).values
    x_data = data.drop(["Label","Circuit"], axis=1)




    scaler = MinMaxScaler(feature_range=(0, 1))
    x_data = scaler.fit_transform(x_data)

   # plot the correlated features
    sns.heatmap(
        corr_matrix,
        vmin=-1, vmax=1, center=0,
        cmap=sns.diverging_palette(20, 220, n=200),
        square=False
    )
    plt.title("Features correlation")
    plt.savefig("Features correlation.png",dpi=300, bbox_inches='tight')
```

```
    plt.show()



    # plot the correlated features
    sns.heatmap(
        corr_matrix_update,
        vmin=-1, vmax=1, center=0,
        cmap=sns.diverging_palette(20, 220, n=200),
        square=False
    )
    plt.title("Features correlation after drop")
    plt.savefig("Features correlation after drop.png")

    plt.show()
    return x_data, y_label
def fit_random_forest_classifier(X, y):
    '''
    INPUT: names are pretty self explanatory
    OUTPUT: none - prints the confusion matrix and accuracy
    '''

    ## validation data for testing
```

```
from sklearn.ensemble import RandomdomForestClassifier




#First let's create training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
print(len(X_test),"Test ?? train :",len(X_train))
#We could grid search and tune, but let's just fit a simple model to
#instantiate
clf = RandomForestClassifier(n_estimators=100, max_depth=None, min_sa

#fit
clf.fit(X_train, y_train)

#predict
y_preds = clf.predict(X_test)

#score
print(confusion_matrix(y_test, y_preds))
acc = accuracy_score(y_test, y_preds)

# print("y_label :", type(y), y)
```

```
    F_measure=f1_score(y_test, y_preds, average='macro')
    precision=precision_score (y_test, y_preds, average='macro')



    print("F_measure",F_measure,"precision",precision)
    return acc

    from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.utils import shuffle

def fit_random_forest_classifier(X, y):
    '''
    INPUT: names are pretty self explanatory
    OUTPUT: none - prints the confusion matrix and accuracy
    '''
    #First let's create training and testing data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=

    #We could grid search and tune, but let's just fit a simple model to
    #instantiate
```

```
    clf = RandomForestClassifier(n_estimators=100, max_depth=None)

    #fit
    clf.fit(X_train, y_train)

    #predict
    y_preds = clf.predict(X_test)

    #score
    print(confusion_matrix(y_test, y_preds))
    acc = accuracy_score(y_test, y_preds)
    print(acc)
    return acc


def do_pca(n_components, data):
    '''
    Transforms data using PCA to create n_components, and provides back
    transformation.

    INPUT: n_components - int - the number of principal components to cr
           data - the data you would like to transform

    OUTPUT: pca - the pca object created after fitting the data
```

```
        X_pca - the transformed X matrix with new number of compone
    '''
    X = StandardScaler().fit_transform(data)
    pca = PCA(n_components)
    X_pca = pca.fit_transform(X)
    return pca, X_pca



if "__name"=="__main__":
    X=prepare_data("Benchmark_Feature_Extraction.xlsx")
    X,y=preprocess_data (X)
    fit_random_forest_classifier(X, y)
```

# APPENDIX C:
# FEATURE EXTRACTION TCL SCRIPT

```
####################################################
# GENUS RTL Compiler Script for Synthesis
####################################################

#*******************************************
# SETUP
#*******************************************
#-------------------------------------------------------------
# Set up variables for this design
#-------------------------------------------------------------

set DESIGN_NAME      {Name_of_the_Design}

# Search path for library files (liberty file)
```

```
echo "LIBRARY_SEARCH_PATH "

set LIBRARY_SEARCH_PATH
{<Path to the libary to the technolgy used to sysnthesis, i.e.GPDK045}

echo "LIBRARY_NAMES  "

set LIBRARY_NAMES   {fast_vdd1v0_basicCells.lib }

# Path to Verilog design files

set HDL_SEARCH_PATH   {<Path to the top RTL design>}

set SYN_FILE_DIRECTORY  {<Path to the to the synthesis directory >}

set HDL_FILENAMES  {<Name of the top design>}

set UNGROUP_INSTANCE_FULLNAMES {}

set POWER_ANALYSIS  true

set FLATTEN    True
```

```
#source ./IMPORT/setup.tcl

#----------------------------------------------------------------
# Set application variables based on setup
#----------------------------------------------------------------
echo "Set application variables based on setup"

set_db init_lib_search_path $LIBRARY_SEARCH_PATH

set_db library $LIBRARY_NAMES

set_db hdl_search_path $HDL_SEARCH_PATH

set_db hdl_track_filename_row_col $POWER_ANALYSIS

#Debug verbosity from 0 to 9

set_db information_level 9

#*******************************************
# LOAD DESIGN
#*******************************************
echo "Load"
```

```
read_hdl $HDL_FILENAMES

elaborate

#dc::current_design $DESIGN_NAME

# Check for unresolved refs & empty modules

#check_design -unresolved


#*********************************************
# Setting the clock period & clock Name
#********************************************

#read_sdc ./${DESIGN_NAME}.premapped.sdc

#clock period in ps

set CLK_PERIOD 20000

set CLK_PORT_NAME clk

set CLK_NAME 50MHz
```

```
set clock [define_clock -period $CLK_PERIOD -name $CLK_NAME [clock_ports

set_input_delay -clock $CLK_NAME 0 [all_inputs]

set_output_delay -clock $CLK_NAME 0 [all_outputs]

#*****************************************
# COMPILE
#*****************************************

# Perform logic synthesis: technology mapping + logic optimization

syn_generic

syn_map

syn_opt

#*****************************************
# GENERATE REPORTS
#*****************************************
```

```
# Output some useful results of synthesis

report_timing > ./output/synth_report_timing.txt

report_gates > ./output/synth_report_gates.txt

report_power > ./output/synth_report_power.txt

report_area > ./output/synth_report_area.txt

#*******************************************
# SAVE DESIGN
#*******************************************
# Write gate-level RTL

write_hdl > ./output/${DESIGN_NAME}.mapped.v

# Write SDF (standard delay format) for functional simulations using tim
write_sdf > ./output/${DESIGN_NAME}.mapped.sdf

gui_show
# Write design constraints in SDC and another format
write_sdc > ./output/${DESIGN_NAME}.mapped.sdc
```

```
write_script > ./output/${DESIGN_NAME}.mapped.g
```

# APPENDIX D:

# FIGURES

```
==============================================================
  Generated by:            Genus(TM) Synthesis Solution 18.14-s037_1
  Generated on:            Sep 05 2022  04:53:35 pm
  Module:                  aes_128
  Operating conditions:    PVT_1P1V_0C (balanced_tree)
  Wireload mode:           enclosed
  Area mode:               timing library
  Description:             AES_100 (Trojan Free)
==============================================================


  Gate   Instances    Area       Library
--------------------------------------------
DFFQXL         128   700.416    fast_vdd1v0
INVXL          128    87.552    fast_vdd1v0
SDFFQX1        128   963.072    fast_vdd1v0
--------------------------------------------
total          384  1751.040


     Type      Instances   Area   Area %
-----------------------------------------
sequential           256 1663.488   95.0
inverter             128   87.552    5.0
unresolved            20    0.000    0.0
physical_cells         0    0.000    0.0
-----------------------------------------
total                404 1751.040  100.0
```

**Figure D.1: Synthesis output area report for AES-T100 Trojan free**

```
================================================================
  Generated by:          Genus(TM) Synthesis Solution 18.14-s037_1
  Generated on:          Oct 30 2022  08:09:11 pm
  Module:                top
  Operating conditions:  PVT_1P1V_0C (balanced_tree)
  Wireload mode:         enclosed
  Area mode:             timing library
  Description:           AES_100 (Trojan Infected)
================================================================

                              Leakage      Dynamic        Total
     Instance          Cells Power(nW)   Power(nW)     Power(nW)
----------------------------------------------------------------
top                   184858 18560.028 11853063.392 11871623.421
  AES_rf_S4_1_S_0        494    47.474    22572.666    22620.140
  AES_rf_S4_2_S_1        494    47.474    22572.666    22620.140
  AES_rf_S4_3_S_2        494    47.474    22572.666    22620.140
  AES_rf_S4_2_S_0        494    47.474    22482.853    22530.327
  AES_rf_S4_3_S_1        494    47.474    22572.666    22620.140
  AES_rf_S4_4_S_2        494    47.474    22572.666    22620.140
  AES_rf_S4_3_S_0        494    47.474    22482.853    22530.327
  AES_rf_S4_4_S_1        494    47.474    22572.666    22620.140
  AES_rf_S4_1_S_2        494    47.474    22572.666    22620.140
  AES_rf_S4_4_S_0        494    47.474    22482.853    22530.327
  AES_rf_S4_2_S_2        494    47.474    22572.666    22620.140
  AES_r7_t0_t3_s4        498    47.456    21375.273    21422.729
  AES_r5_t2_t1_s4        498    47.456    21375.273    21422.729
  AES_r6_t0_t0_s4        498    47.456    21375.273    21422.729
  AES_r6_t2_t0_s4        498    47.456    21375.273    21422.729
  AES_r6_t3_t0_s4        498    47.456    21375.273    21422.729
  AES_r3_t2_t1_s4        498    47.456    21375.273    21422.729
  AES_r9_t2_t0_s4        498    47.456    21375.273    21422.729
  AES_r4_t1_t2_s4        498    47.181    21296.682    21343.862
  AES_r4_t3_t2_s4        498    47.181    21057.174    21104.355
  AES_r4_t0_t2_s4        498    47.181    21296.682    21343.862
  AES_r2_t1_t2_s4        498    47.181    21296.682    21343.862
```

**Figure D.2: Synthesis output power report for AES-T100 Trojan Infected**

```
=============================================================
  Generated by:           Genus(TM) Synthesis Solution 18.14-s037_1
  Generated on:           Jan 21 2023  09:56:50 pm
  Module:                 aes_128
  Operating conditions:   PVT_1P1V_0C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
=============================================================


  Gate   Instances    Area        Library
----------------------------------------------
DFFQXL        2256  12344.832    fast_vdd1v0
INVX1          320    218.880    fast_vdd1v0
INVXL          768    525.312    fast_vdd1v0
MX2XL           32     76.608    fast_vdd1v0
MXI2XL        1152   2757.888    fast_vdd1v0
SDFFQX1        432   3250.368    fast_vdd1v0
XNOR2X1        752   1800.288    fast_vdd1v0
----------------------------------------------
total         5712  20974.176


     Type      Instances     Area   Area %
-------------------------------------------
sequential          2688 15595.200   74.4
inverter            1088   744.192    3.5
unresolved            20     0.000    0.0
logic               1936  4634.784   22.1
physical_cells         0     0.000    0.0
-------------------------------------------
total               5732 20974.176  100.0
```

**Figure D.3: Synthesis output gate report for AES-T1000 Trojan Free**

```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 18.14-s037_1
  Generated on:           Jan 21 2023  09:56:50 pm
  Module:                 aes_128
  Operating conditions:   PVT_1P1V_0C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================


Path 1: MET (19829 ps) Setup Check with Pin a10/k3a_reg[29]/CK->D
     Startpoint: (R) a9/out_1_reg[93]/CK
          Clock: (R) 50MHz
       Endpoint: (R) a10/k3a_reg[29]/D
          Clock: (R) 50MHz

                    Capture       Launch
        Clock Edge:+   20000           0
       Src Latency:+       0           0
       Net Latency:+       0 (I)       0 (I)
           Arrival:=   20000           0

             Setup:-      18
     Required Time:=   19982
      Launch Clock:-       0
         Data Path:-     152
             Slack:=   19829

#---------------------------------------------------------------------------------------------
#    Timing Point     Flags   Arc    Edge   Cell      Fanout Load Trans Delay Arrival Instance
#                                                            (fF)  (ps)  (ps)   (ps)  Location
#---------------------------------------------------------------------------------------------
  a9/out_1_reg[93]/CK -       -      R     (arrival)   2708    -     0     -       0    (-,-)
  a9/out_1_reg[93]/Q  -       CK->Q F      DFFQXL         1  0.3     6    44      44    (-,-)
  a10/g4046/Y         -       A->Y  R      XNOR2X1        2  0.9    10    43      87    (-,-)
  a10/g3939/Y         -       B->Y  F      XNOR2X1        2  0.7     9    32     118    (-,-)
  a10/g3826/Y         -       B->Y  R      XNOR2X1        1  0.3     6    34     152    (-,-)
```

**Figure D.4: Synthesis output Timing report for AES-T1000 Trojan Free**

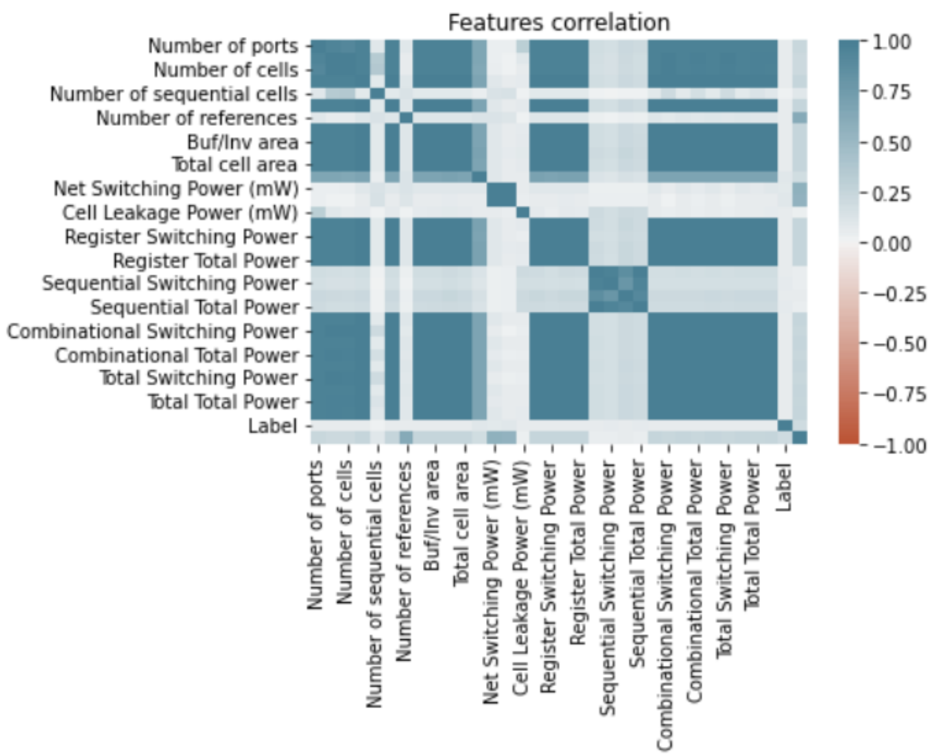


**Figure D.5: Correlation coefficients between features.**

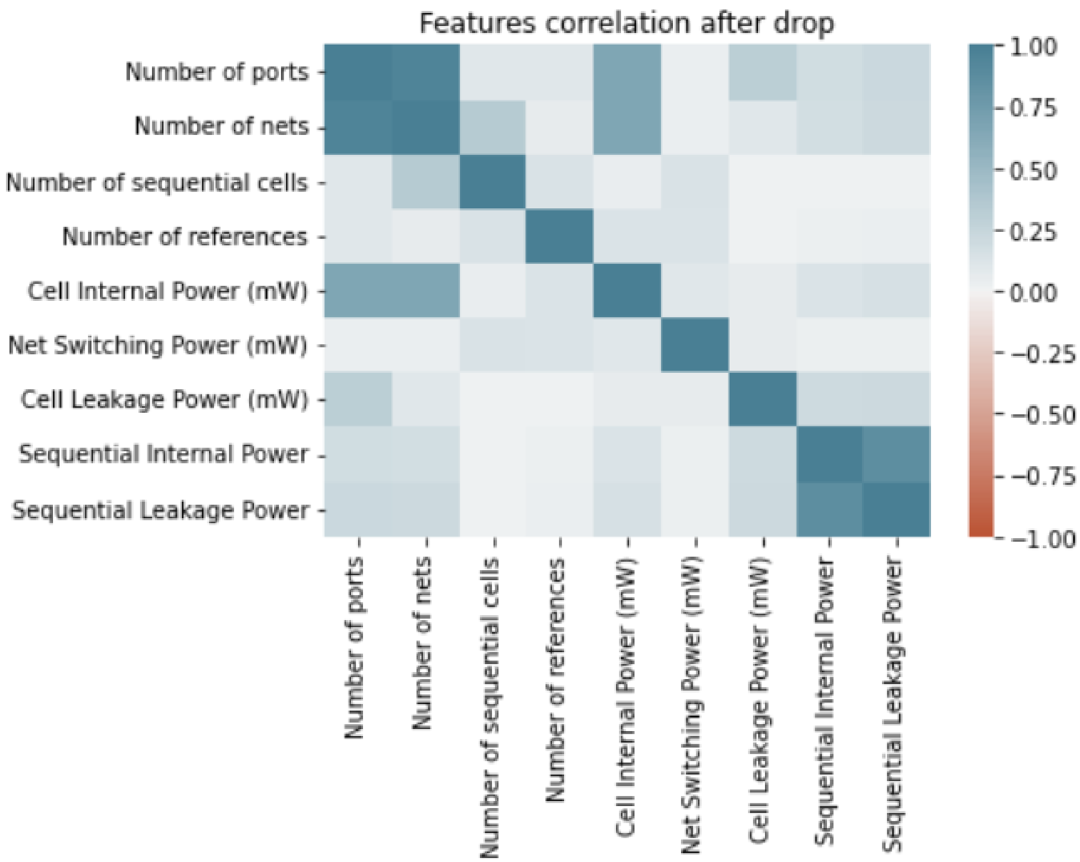


**Figure D.6: Correlation coefficient between features after dropping.**

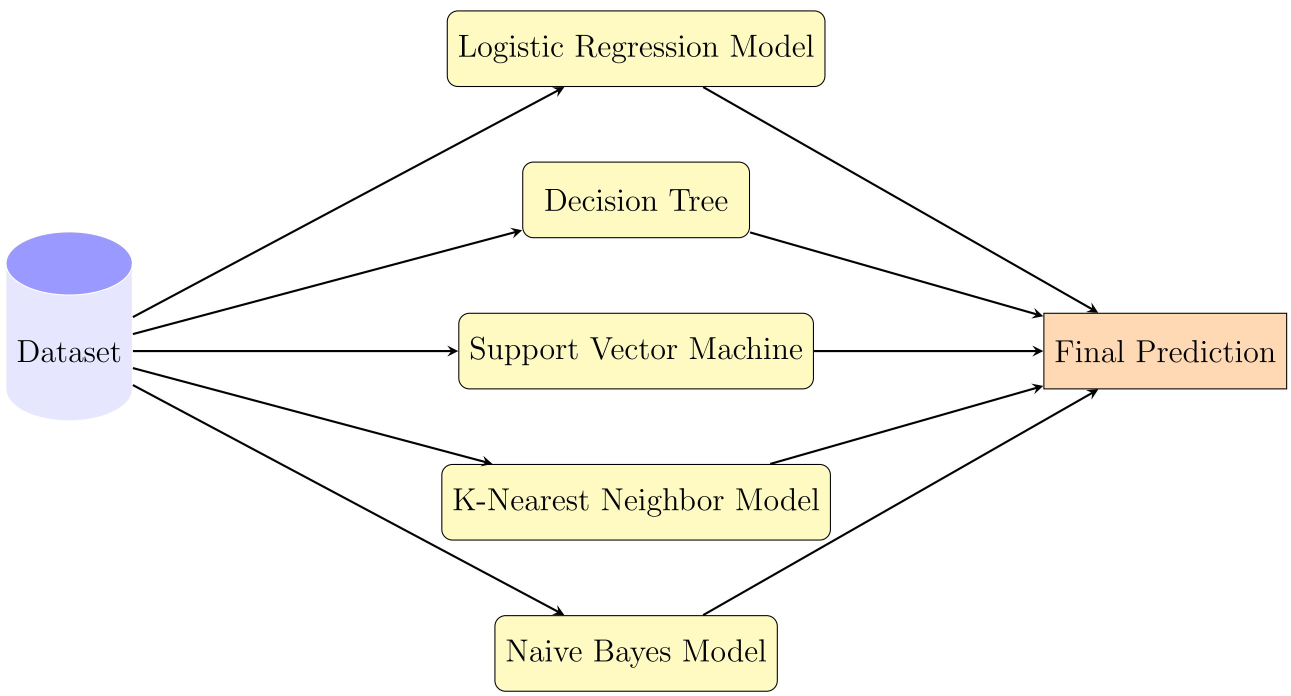


**Figure D.7: Hybrid Ensemble Model**