ML-ASSISTED SIDE CHANNEL SECURITY APPROACHES FOR HARDWARE TROJAN DETECTION AND PUF MODELING ATTACKS

A thesis submitted in partial fulfillment of
the requirements for the degree of
Master of Science in Computer Engineering

By

NIRAJ PRASAD BHATTA
M.Tech., Jain University, India, 2017
B.E., Anna University, India, 2014

2024
Wright State University

WRIGHT STATE UNIVERSITY
College of Graduate Programs and Honors Studies

April 19, 2024

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Niraj Prasad Bhatta ENTITLED ML-Assisted Side Channel Security Approaches for Hardware Trojan Detection and PUF Modeling attacks BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Computer Engineering.

_____
Fathi Amsaad, Ph. D.
Thesis Director

_____
Thomas Wischgoll, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination

_____
Fathi Amsaad, Ph.D.

_____
Wen Zhang, Ph.D.

_____
Kenneth Hopkinson, Ph.D.

_____
Paula Bubulya, Ph.D.
Interim Dean, College of Graduate Programs & Honors Studies

Abstract

Bhatta, Niraj Prasad. M.S.C.E., Department of Computer Science and Engineering, Wright State University, 2024. ML-Assisted Side Channel Security Approaches for Hardware Trojan Detection and PUF Modeling attacks.

Hardware components are becoming prone to threats with increasing technological advances. Malicious modifications to such components are increasing and are known as hardware Trojans. Traditional approaches rely on functional assessments and are not sufficient to detect such malicious actions of Trojans. Machine learning (ML) assisted techniques play a vital role in the overall detection and improvement of Trojan. Our novel approach using various ML models brings an improvement in hardware Trojan identification with power signal side channel analysis. This study brings a paradigm shift in the improvement of Trojan detection in integrated circuits (ICs).

In addition to this, our further analysis towards hardware authentication extends towards PUFs (Physical Unclonable Functions). Arbiter PUFs were chosen for this purpose. These are also Vulnerable towards ML attacks. Advanced ML assisted techniques predict the responses and perform attacks which leads to the integrity of PUFs. Our study helps improve ML-assisted hardware authentication for ML attacks. In addition, our study also focused on the defense part with the addition of noise and applying the same attack ML-assisted model.

Detection of Trojan in hardware components is achieved by implementing machine.

learning techniques.for this Purpose several Machine learning models were chosen. Among them, Random forest classifier(RFC) and Deep neural network shows the highest accuracy. This analysis plays a vital role in the security aspect of all hardware components and sets a benchmark for the overall security aspects of hardware. Feature extraction process plays major role for the improvement of accuracy and reliability of hardware Trojan classification.

Overall, this study brings significant improvement in the field of overall hardware security. Our study shows that RFC performs best in hardware classification with an average of 98. 33% precision of all chips, and deep learning techniques give 93. 16% precision of all chips. Moreover, on the hardware authentication side, RFC performs the best of all other models with the accuracy of 89% in the attack part and 81% in the defense part in 6000 data samples.

Contents

List of Figures

List of Tables

Acknowledgment

Firstly, I would like to thank the Department of Computer Science and Engineering at Wright State University for providing me with such a beautiful academic environment and for fostering a culture of innovation and collaboration. The opportunity to work alongside talented peers and dedicated faculty members has been invaluable to my growth as a researcher.

I express my deepest gratitude to our supervisor Dr. Fathi Amsaad for his invaluable guidance, unwavering support, and insightful feedback throughout the entire process of this thesis. His expertise and encouragement have been instrumental in shaping this work.

I am deeply grateful to my parents and all members of the family for their constant love, encouragement, and understanding during this journey. Their belief in me has been a source of strength and motivation.

I am also deeply grateful to all my colleagues from our SMART Lab.,Wright state university, and all who know me and have supported me in either way. Your encouragement and belief in me have been invaluable.

Lastly, I extend my heartfelt thanks to all individuals who contributed their time and insights to this study.

This thesis would not have been possible without the support and encouragement of all those mentioned above. Thank you.

1 Introduction

1.1 The Importance of Hardware Security

Today, with digital technologies becoming an important part of nearly every aspect of our life, the need to protect hardware systems has become more crucial than ever. It sets up the critical framework essential for establishing and securing global faith in technological innovations. This section aims to shed light on the importance of hardware security, focusing on its important role in ensuring the security of individual devices and the comprehensive systems that support the security framework of individuals and the national security infrastructures[1].

Hardware security is the cornerstone assuring the stability and durability of our digital environments. Bridging from the microchips at the heart of our smartphones to the extensive networks that bolster indispensable infrastructure,the reliability of hardware parts is the first line of protection against a myriad of cyber threats and antagonistic entities. Hardware security is very essential for making overall digital systems more secure and reliable.There is a need of security in many things, starting from personal items to general security including national security concerns.

Additionally, as the technology is improving day by day, there is need for security for all the hardware components, since privacy is the major concern for all.People trust more on the digital systems considering it's basic architecture that provides security in concern with the illegal access. In the similar way, all the sectors from individual,organisations as well as government relies on secure hardware components.they

are fully dependent on the digital platforms to protect their sovereignty to protect against different kinds of ongoing cyber attacks which are risky for national security.

In general, the robustness of hardware devices serves as the backbone for security within the digital era, maintaining the framework for our interconnected world. By understanding and addressing the importance of hardware security, we can improve our shared potential to withstand upcoming threats, paving the direction against future threats and setting the groundwork for a safer, more secure technological world for decades to come.

1.2 Hardware Trojans and Power Side-Channel Analysis

Hardware Trojans, commonly known as hardware implants or backdoors, are harmful modifications that affect the framework or manufacturing procedure of integrated circuits (ICs) that are the key components of modern electronic systems. Such Trojans constitute a significant risk to the performance of systems by degrading the functionality, security, and dependability of electronic systems.The reliability of hardware parts can be defined mathematically with the exponential reliability function:

$$R(t) = e^{-\lambda t} \tag{1.1}$$

where $R(t)$ is the reliability at time t, λ is the failure rate of the component, and t is the time[2, 3].

This technique serves to predict the lifetime and failure instances of hardware parts, ensuring that they stay safe and functional throughout time. Hardware Trojans exist hidden within the actual framework of the device itself, making them extremely difficult to identify and mitigate. The idea of hardware Trojans covers a wide range of harmful behaviors, including unauthorized data leakage, modification of system

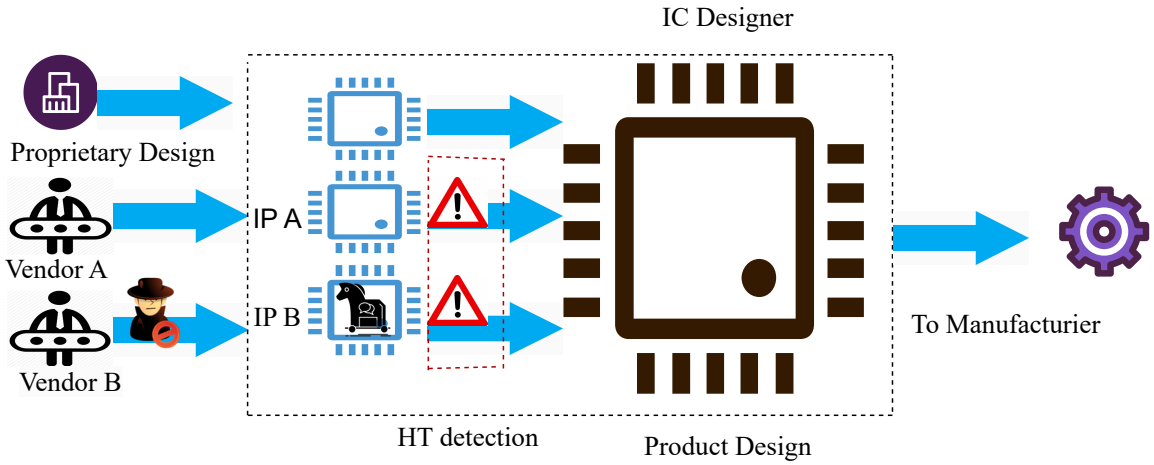

Figure 1.1: HT Detection Scenario in IC Design Process

performance, as well as total system takeover. These unfavorable modifications may be introduced at various phases of the IC life cycle, across the design stage to the fabrication, assembly, as well as verification stages. Attackers might take advantage of weak stages in the supply chain to hide these kinds of Trojans, either by introducing malicious components throughout the design phase or by changing the process of production in a manner that remains undiscovered[4]. Fig [1.1] below shows IC design process, where a designer integrates IP blocks A and B from vendors A and B into product. hardware Trojan detection, shown with the red circle, enables her to quickly recognize any introduced Trojans, for example one possibly buried in IP B.

Conventional approaches for hardware detection Trojans often rely on functional evaluation, a process that involves introducing the device to a number of inputs and analyzing its outputs in order to identify any deviations from the predicted behavior.However, this method has its limitations as it may not detect insignificant Trojans that do not impact the normal functioning of the device or only operate in certain conditions.

The analysis of power side-channel signals has transformed into a highly effective method for the identification of hardware Trojans, indicating improved accuracy

and efficacy. This method is especially helpful for complicated circuits crucial for important areas, which includes critical infrastructure, aerospace, military, and consumer electronics.With the increased sophistication of hardware Trojans and the increasing complexity of current electronic systems, novel detection approaches such as power side-channel analysis are crucial for ensuring the integrity and security of hardware components and defending against emerging threats[5].

Information entropy for data security is deeply connected with Hardware Trojans and Power Side-Channel Analysis as it measures the unpredictability of cryptographic keys, important in detecting anomalous patterns and vulnerabilities created by Trojans through deviations in power consumption signals.Information entropy is a measure of unpredictability or randomness, calculated for systems to understand their complexity and security level[5, 6, 7]. The formula for information entropy is given by:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_b P(x_i) \tag{1.2}$$

where:

- $H(X)$ is the entropy of a random variable X with possible values $x_1, x_2, \ldots, x_n$,
- $P(x_i)$ is the probability of x_i,
- b is the base of the logarithm used, commonly 2 for binary systems[8].

1.3 Arbiter PUFs and Machine Learning Vulnerabilities

Arbiter Physical Unclonable Functions (PUFs) are a hardware authentication mechanism widely applied to provide electronic systems with unique and unreplicable identities. By utilizing the inherent inconsistencies found in manufacturing processes,

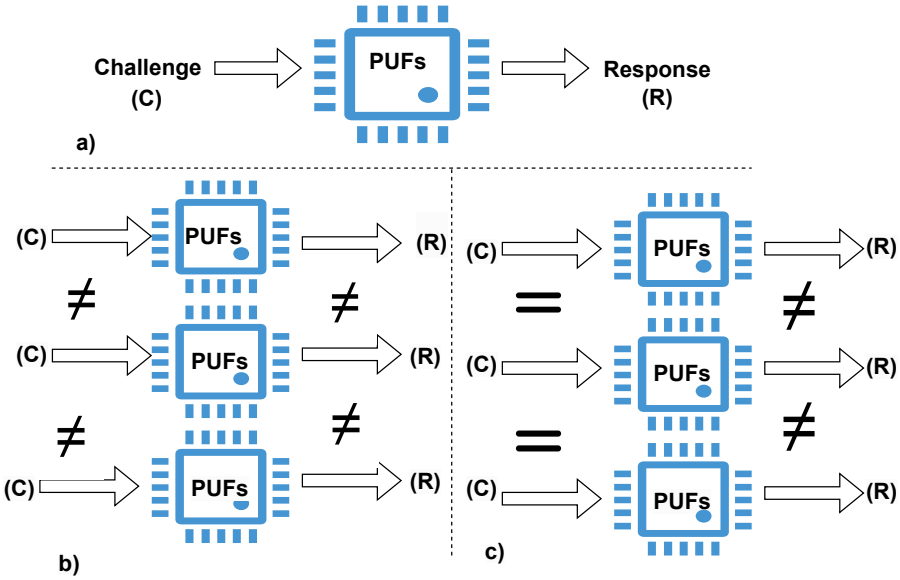

Figure 1.2: PUF concept: (a) Challenge-Response Pairs (CRPs), (b) Features where a PUF provides varying responses to distinct challenges, and (c) Attributes generating unique responses among different PUFs due to process inconsistencies

they generate unique responses to particular questions, functioning as a hardware-based identification system. These different responses are crucial in establishing a device's authenticity or in building up secure channels for communication between devices as shown in Fig [1.2].The mathematical expression that relates challenge and response for a PUF is as follows:

$$R = f(C) \tag{1.3}$$

where, R is the response, C is the challenge, and f is the PUF's precise function that connects both R and C [9].

Although, arbiter PUFs are widely used and are more secure are still vulnerable. Certain kinds of attacks tend to minor changes in the arbiter PUF responses that leads to predict new responses by new challenges. Such kinds of attacks use advanced algorithms with massive dataset for its operation that can replicate the behavior of arbiter PUFs.

The vulnerable nature of Arbiter PUFs to ML-based attacks requires an in-depth analysis of their security features and the remedies that address this vulnerability. This overview includes learning Arbiter PUFs' necessary mechanisms, a thorough study into the nature of ML attacks, and the selection of acceptable defensive techniques.

One potential method to mitigate this security risk is by increasing the randomness and unpredictability of PUF responses by using techniques such hardware obfuscation, the addition of noise, or changes of challenge-response pairs. These techniques are designed to improve the complexity of PUF responses, thus reducing ML algorithms potential to accurately predict and reduce the probability of effectiveness. Another method is to include extra levels of protection, for example cryptographic protocols, to improve on the authentication given by Arbiter PUFs. By integrating multiple security mechanisms, designers may develop stronger and more capable systems that are less prone to exploitation by smart attacks.

The equation for predicting responses to new challenges applying a ML model can be written as:

$$\hat{R} = \hat{f}(C_{\text{new}}) \tag{1.4}$$

This expression shows the expected responses $\hat{R}$ to a new challenge C_{new} using ML framework $\hat{f}$ that correlates the behavior of the PUF. The entropy, $H(R)$, of the responses can be written as:

$$H(R) = -\sum_{i=1}^{n} P(r_i) \log_2 P(r_i) \tag{1.5}$$

where $H(R)$ defines the unpredictability of the response set. Here, $P(r_i)$ is the probability for every unique response r_i, and the result is over all potential responses n. Increasing this entropy causes it to be more difficult for ML models to predict new responses in a correct way.

Furthermore, current efforts are currently focused upon creating unique PUF designs and methods of authentication which provide improved security features and robustness to machine learning-based attacks. This consists of exploring various PUF architectures, the study of unique physical principles for manufacturing one-of-a-kind IDs, and the use of cryptographic advancements for improving hardware authentication mechanisms. Adding noise to the response or the challenge can be mathematically described as:

$$R' = f(C + N_C) + N_R \tag{1.6}$$

where N_C indicates noise introduced to the challenge C, N_R denotes noise added directly to the response, and R' is the updated response. This approach seeks to improve the system's level of entropy and, thus, its defense against attacks.

Arbiter PUFs are essential for the safety of electronic systems, their exposure to machine learning attacks underlines the necessity of ongoing evaluation and innovation within hardware security. Addressing these susceptibilities and implementing effective strategies would allow designers and researchers to improve hardware authentication techniques and limit the chance of new possible risks.

1.4 Problem Statement

The challenges posed by hardware Trojans and the vulnerability of Arbiter Physical Unclonable Functions (PUFs) to machine learning-based attacks underline the need for improvement in the realm of hardware security. This subsection aims to articulate the specific problems that the thesis seeks to address within this framework.

- **Complexity of Detecting Hardware Trojans** : Conventional methodologies for finding hardware Trojans generally focus on functional evaluation, where predetermined patterns are executed and the device's behavior are observed.

However, hardware Trojans may be engineered to avoid detection during functional testing. it is possible by staying dormant or activating only under specified situations.This complexity in detection is a major problem as it enables Trojans to avoid traditional security processes and harm system security. Consequently, there is an urgent need to create more robust and effective detection techniques capable of recognizing hardware Trojans, even when they are supposed to be hidden.

- **Vulnerabilities of Arbiter PUFs to Machine Learning Attacks** : Arbiter PUFs are extensively applied for hardware authentication, using the unavoidable variations in manufacturing to provide different IDs. Yet, recent research have demonstrated that Arbiter PUFs are prone to attacks employing machine learning. these are capable of precisely anticipating PUF responses and therefore breaching their security. This weakness impairs the dependability and trustworthiness of Arbiter PUFs as authentication methods underlining the demand for enhanced security measures to defend against such vulnerabilities.

By addressing these challenges, this thesis seeks to contribute to the improvement of hardware security by providing improved detection techniques for hardware Trojans and strengthening the robustness of Arbiter PUFs towards machine learning attacks. This requires doing in-depth research to understand the underlying shortcomings and restricts of current security methods. also, it helps in creating and implementing innovative techniques to mitigate these vulnerabilities and increase the security of electronic systems. Through these efforts, the thesis intends to increase the trustworthiness and integrity of hardware components and boost overall system security and evolving threats.

1.5 Research Objectives

The main goal of this thesis is to elevate the field of cybersecurity through two essential contributions: first, by improving the detection of hardware Trojans, a critical aspect of safeguarding digital ecosystems from covertly embedded malware within hardware components; and second, by advancing the defense mechanisms of PUFs, specifically Arbiter PUFs, which are fundamental to cryptographic security through their provision of unique,identifiers critical for authentication processes. This research aims to:

- Conduct an analysis of existing hardware Trojan detection methodologies, pinpointing their limitations and exploiting innovative technologies to devise more accurate, efficient, and scalable detection strategies.
- Investigate the susceptibility of Arbiter PUFs to Machine learning attacks, evaluating their current defense mechanisms, and proposing novel solutions that enhance their robustness and reliability as a cryptographic tool.

Through these efforts, this thesis seeks not only to push the boundaries of what is currently achievable in cybersecurity but also to lay the groundwork for future research that could further digital security protocols and infrastructures. To meet these objectives, the research will delve into a comprehensive examination of current Trojan detection methodologies to identify and address their shortcomings. By leveraging cutting-edge technology and innovative approaches, we aspire to develop more effective and efficient methods for identifying Trojans, thus minimizing their threat.

This research will explore the weaknesses of PUFs to advanced attacks, with the goal of identifying new advancements that will ensure their ongoing reliability in

cryptographic security.The objective of this research is to improve cybersecurity and set new benchmarks for digital protection by methodically examining key areas for enhancing cryptographic protocols and overall digital security.

1.6 Scope and Limitations

The project aims to advance cybersecurity by focusing on two key areas: Improving Trojan detection and strengthening the defense mechanisms in the PUFs. Trojans are discreetly integrated into authentic programs, present a significant risk to the integrity of digital frameworks. our research will meticulously dissect Machine learning strategies for identifying Trojans. In addition, our main emphasis is on strengthening Physically Unclonable Functions (PUFs), which are the fundamental building blocks of cryptographic security. PUFs are widely acknowledged for their distinct and inviolable IDs that are essential for authentication purposes. In order to achieve these goals, our study will examine existing methods for detecting Trojans, identify any shortcomings, and propose innovative solutions by developing more efficient and effective approaches to mitigate their risk.

We will also explore the susceptibility of PUFs to very sophisticated attacks. PUFs, while their crucial function in safeguarding digital communications and information, have inherent weaknesses. The objective of this study is to investigate these weaknesses and improve approaches and technologies to strengthen PUFs' capacity to withstand attacks, therefore assuring their dependability in cryptographic defense.

Through a thorough examination of these crucial areas, the project aims to surpass current cybersecurity standards, therefore making a substantial influence on the domains of cybersecurity, digital safety, and cryptographic approaches. Overcoming these hurdles, we want to build new norms of digital security and safety, leading to a more secure and trustworthy digital world.

1.7 Methodology Overview

This research discusses comparative analytical approach undertaken in the evaluation of the performance of various machine learning algorithms. Selection of these algorithms have been very chosen very meticulously, taking care that they ensure a very robust framework with myriad components of the algorithmic capability and applicability.

First, the Selection of an algorithm was more on the relevance of the domain and acceptance in similar widespread studies. Further considerations were made in view of the scalability, interpretability, and computational efficiency of the algorithm, since these characteristics play essential roles in real-world deployment scenarios. Moreover, the analysis would be impartial towards all the algorithms, ensuring equal consideration for the performance of each algorithm in its attitude.This involves rigorous data preparation approaches to address problems like as missing values, data standardization, and feature selection to increase the quality of input data. Additionally, the dataset was partitioned into training, validation, and testing sets using proper procedures to verify the generalizability of findings[10].

Model evaluation metrics were selected to measure each such algorithm was effective, in line with the goals of the study. Furthermore, model assessment measures were carefully designed to evaluate the success of each algorithm in addressing the goals of the research. These measures contain a broad variety of performance indicators including as accuracy, precision, recall and F1-score offering a complete insight of algorithmic performance across many dimensions[11].

Overall, the comparative analytical approaches implemented in this work intends to give significant insights into the strengths and shortcomings of different machine learning algorithms, in order to contribute toward making the selection and implementation of algorithms in real-world applications easier.

1.8 Significance of the Study

The results and findings of this work have immense significance for the hardware security, providing essential insights that might considerably effect the development of hardware design and the creation of security solutions. Through an in-depth examination of how machine learning algorithms function in detecting and addressing security concerns, this study not just broadens our knowledge of current security approaches but also sets the framework for building stronger and dynamic hardware security mechanisms.

The insights obtained from this research lies in its capacity to determine the design and deployment of active security approaches within hardware systems. By using machine learning to discover vulnerabilities and possible attacks, hardware designers may build preemptive security solutions into the architecture, hence enhancing the resistance of the system against developing attacks. Moreover, the insights acquired through this work may lead to advancements in security procedures for hardware devices, notably in fields such as identifying anomalies, intrusion avoidance, and attack mitigation.By leveraging the capabilities of machine learning algorithms to identify minor patterns and abnormalities in system behavior, security protocols may be strengthened to enable real-time monitoring and adaptive response mechanisms, thereby mitigating the risks posed by sophisticated cyber attacks.

Moreover, the potential of this study to inspire cross-disciplinary collaboration among hardware engineers, cybersecurity specialists, and machine learning experts cannot be overstated. This creative synergy promises to generate innovative solutions that blend the latest in hardware engineering with cutting-edge machine learning approaches, culminating in the production of highly secure and durable hardware systems.

Overall, the consequences of this study each far beyond academic exploration,delivering substantial benefits for the larger field of hardware security. By leveraging the insights gained from this research, stakeholders are well-positioned to pioneer the next wave of hardware systems,that are not only safe by design but also adaptable and robust in the response of increasing cyber threats.

1.9 Contribution

The gaps in existing works on ML models for hardware security shows the need for a improved comparative study of advanced ML techniques for classification. Such a review is essential to our research for the complexities of detecting hardware Trojans and protecting PUFs, specifically considering the increasing nature of attacks and the limitations of current approaches. This study is meant to evaluate the effectiveness, efficiency, and application of various ML approaches in addressing the complicated issues of hardware security, thereby contributing considerably towards understanding and addressing prior work's limitations[12].

- **Application of ML for Hardware Trojan Detection**: Our Study explores the current improvements in machine learning methods that significantly improve the detection of hardware Trojans, providing a complete evaluation of their effectiveness and efficiency in detecting invisible, malicious changes within hardware components.
- **Protecting PUFs with ML**: Our study explores the role of ML in strengthening the security of PUFs, emphasizing innovative methods that reduce vulnerabilities to ML-based attacks as well as ensure the trustworthiness of these critical security features.
- **Multidisciplinary Strategies to Hardware Security**: Our study shows the importance of collaborative efforts throughout cybersecurity, hardware engineering,

and machine learning fields to address complicated challenges in hardware security for creating robust defense mechanisms.

1.10 Thesis Structure

This Portion provides the general introduction by presenting a concise and clear outline of the thesis. this will make readers to be clear with flow of the contents, thus providing a summary of the study carried out. This study includes multiple chapters, each dedicated to various aspects of hardware security, from theoretical foundations to practical implementations and novel research findings[13, 5].

- **Chapter 2: Background** gets the evolution of hardware security, outlining significant findings and improvements in technology. It thoroughly explores power side-channel analysis, its methodology, applications, and the significance of ML in detecting hardware Trojans. Furthermore, this chapter explores the idea of PUFs and their importance, while exploring the weaknesses and defenses from a ML perspective. It ends by analyzing current literature, highlighting research gaps, and emerging opportunities.

- **Chapter 3: Methodology for Detecting Hardware Trojans** outlines the hands-on framework designed for the current study. It describes the data collection processes, analysis methods, and proposed framework for detecting hardware Trojans, highlighting the distinct approaches and techniques used to improve hardware detection techniques. In addition to this, this chapter focuses on Securing Hardware Authentication addresses methods to improve hardware authentication mechanisms. It describes the creation of datasets, methodology, and data analysis strategies aimed at strengthening the security of hardware devices from unauthorized access and modification.

- **Chapter 4: Results and Discussion** summarize the findings of the study, providing an in-depth review of the data obtained and the performance of the proposed models. Addresses the implications of these findings within the larger framework of hardware security and the potential consequences for future research and practice.

- **Chapter 5: Conclusion** generates the key ideas obtained from the research, emphasizing the significant contributions of the thesis to the field of hardware security. It focuses on the broad scope and limitations of the study, offering options for future research to better enhance the knowledge and development of secure hardware systems.

- **Chapter 6: Appendix** Provides the code used for the Study

2 Background

2.1 Power Side-Channel Analysis: Techniques and Applications

Power side-channel analysis has grown into a leading framework within the realm of hardware security, notably in the identification of hardware Trojans. This method revolves around monitoring and evaluating the power consumption patterns of a device to find unusual behaviors or abnormalities that might indicate the presence of a Trojan. Below, We delve into the significant modules of power side-channel analysis with suitable case exemplars, and discuss the strengths and limits of these approaches based on current research results[14, 15].

- Basic Techniques of Power Side-Channel Analysis

  Power side-channel analysis exploits the fact that electronic operations consume power and that different operations consume power in subtly different ways. By measuring and analyzing these variations, it's possible to infer the types of operations occurring within a device. The foundational techniques include:

  - Simple Power Analysis (SPA): SPA comprises the direct study of power consumption trends while the device is in operation, seeking to discover cryptographic procedures or other behaviors suggestive of a possible Trojan.

  - Differential Power Analysis (DPA): DPA is a more advanced, incorporating statistical analysis of power across multiple processes to separate particular activities to locate particular activities or find abnormalities that may not be

obvious from individual observations[16].

- Case Studies

  Previous studies shows the use of power side-channel analysis in hardware Trojans detection:

  - Detection in Cryptographic Devices: Recent studies in power side-channel analysis was in identifying Trojans in cryptographic devices. By analyzing power utilization during encryption techniques, researchers could find deviations that showed manipulation, considering that the Trojans were designed to stay inactive until activated by certain triggers.

  - Supply Chain Security: A research effort that focused on supply chain integrity, power side-channel analysis had been used to authenticate that chips created worldwide . This application is essential for military and critical infrastructure industries, where the reliability of hardware components is a key issue[17].

  - IoT Devices: As the use of IoT devices is increasing, power side-channel evaluation becomes essential to assure the security and authenticity of the devices. Experiments were carried out to make sure resource-constrained devices are not compromised by low-power Trojans.

- Strengths and Limitations

  Strengths:

  - Non-Intrusiveness: The evaluation of Power side-channel can be achieved without harming the device, making it appropriate for post-manufacturing inspections and in-field inspections.

  - Reliability Against Stealthy Trojans: This approach is very effective at detecting Trojans built to remain inactive until a certain trigger occurs, as even inactive Trojans can demonstrate low power usage anomalies.

Limitations:

- Complexity and Resource Intensity: Accurate power analysis requires sophisticated equipment and can be resource-intensive, particularly for differential power analysis that requires statistical analysis of large datasets.

- False Positives/Negatives: The precision of power side-channel analysis can be affected by environmental noise and variations in normal device behavior, leading to potential false positives or negatives.

- Evolution of Avoidance methods: with the advancement in the detection techniques, attackers keeps updating certain strategies that fail the detection techniques, including those that have similarities to normal power consumption patterns.

- Recent Research Insights

Previous studies have shown improvement in the sensitivity and accuracy of power side-channel evaluation. this includes design of algorithms that is capable for detecting Trojans even in minor abnormalities. ML techniques has proven to be effective for improving the analysis of power data, with the purpose of decreasing both false positives and false negatives. Nevertheless, the battle remains,as attackers keeps updating new techniques to bypass these new detection methods. overall, although power side-channel study is an essential tool in determining the presence of hardware Trojans, its success depends on ongoing improvement and research efforts to overcome more complex threats. The balance between detection capabilities and the flexible nature of attackers defines the ever-evolving environment of hardware security.

2.2 Machine Learning in Hardware Trojan Detection

The combination of machine learning (ML) models with side-channel analysis for hardware Trojan identification marks a huge leap in cybersecurity. This synergy uses the pattern recognition skills of ML to assess the subtle, often hidden signals within the data received from side-channel investigations such as power usage, electromagnetic emissions, or timing information. Here, we survey several ML models that have been applied for Trojan detection, describe their integration with side-channel analysis, and evaluate their performance and efficiency based on current literature[18].

- Machine learning Models used for Trojan Detection:

  The following are the list of models used for Trojan Detection.

  - Support Vector Machines (SVM): SVMs have been used in Trojan detection for classifications. They are effective in high-dimensional areas, making them useful for evaluating complicated side-channel data.

  - K-Nearest Neighbors (KNN): KNN is employed for its simplicity and effectiveness in classifying applications. It operates by identifying the most comparable data points in the training set to the recently introduced data and provides decisions based on majority voting.KNN classifies a data point based on how its neighbors are classified. It requires no explicit model training phase and makes predictions using a majority vote of its k nearest neighbors[19].

  The Euclidean distance between two points x and x' in an n-dimensional space is given by[20]:

$$d(x, x') = \sqrt{\sum_{i=1}^{n} (x_i - x'_i)^2} \tag{2.1}$$

  The class of a sample is predicted to be the most common class among its k nearest neighbors [5, 21, 22, 23].

-Logistic Regression: This is a basic classification approach that predicts the likelihood of a binary result based on one or more predictor variables. It's highly valued for its simplicity, interpretability, and efficiency in training. While it may not handle complicated structures as well as ensemble or deep learning approaches, it's a decent basic model for binary classification issues. It estimates probabilities using a logistic function.

The logistic or sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

where z is the input to the function.

The model predicts the probability that a given input x belongs to class 1 as follows:

$$P(y = 1|\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.3}$$

Where $P(y = 1|\mathbf{x})$ is the probability that the given input $\mathbf{x}$ belongs to class 1, $\mathbf{w}$ is the weight vector, b is the bias, and $z = \mathbf{w} \cdot \mathbf{x} + b$ [24, 25].

-Random Forest Classifier : The Random Forest Classifier is a robust, ensemble-based technique that pulls together a number of decision trees to generate a more accurate and stable forecast as shown in fig[2.1]. It functions by building an array of trees from randomly chosen subsets of the training data, then combines these predictions to generate a single, more accurate output. Its built-in process for cross-validation via the random selection of data points and features is very successful in preventing overfitting, typically resulting to higher performance in different predicting tasks.The ensemble prediction in a Random Forest model can be expressed as:

$$\hat{y} = \text{mode}\left\{T_1(\mathbf{x}), T_2(\mathbf{x}), \ldots, T_N(\mathbf{x})\right\} \tag{2.4}$$

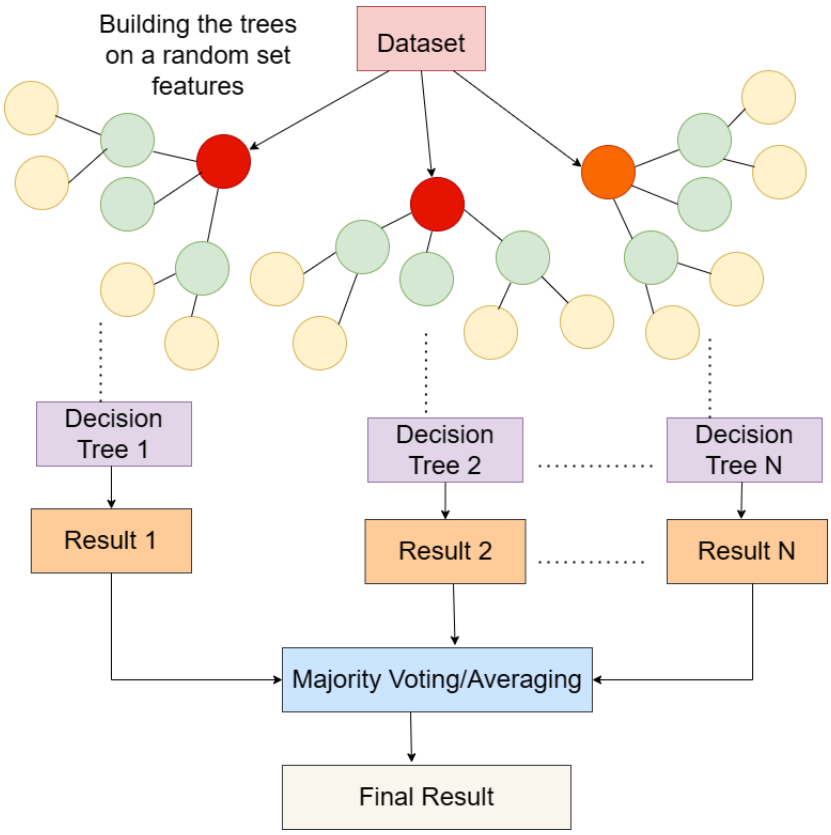

Figure 2.1: Random Forest Classifier model

Where $\hat{y}$ is the predicted outcome, T_i is the i^{th} decision tree's prediction for input $\mathbf{x}$, and N is the number of trees in the forest[26, 27].

- Gradient Boosting Classifier: Gradient Boosting is a sophisticated ensemble approach that creates one tree at a time, with each new tree repairing the faults caused by the preceding ones[28]. The model optimizes for both bias and variance by integrating weak predictive models into a strong learner, resulting in better predicted accuracy. It's usually utilized in cases when performance is crucial and computing resources are adequate for its normally more rigorous training procedure.Gradient Boosting builds models sequentially, each new model correcting errors made by previous ones. Given a loss function L, the model at iteration t, $F_t(x)$, is updated by adding a new weak learner $h_t(x)$ that minimizes

the loss:

$$F_t(x) = F_{t-1}(x) + \rho_t h_t(x) \tag{2.5}$$

where ρ_t is the learning rate, and $h_t(x)$ is chosen to minimize the loss function L [29].

- Ada Boost: Ada Boost is an iterative boosting strategy that modifies the weights of the classifiers at each iteration to reduce mistakes. It successively applies weak classification algorithms to repeatedly updated copies of the data, raising the weight of the observations that were misclassified and lowering it for those that were successfully predicted. This technique leads to a composite model that typically performs better than any single classifier by concentrating on the most problematic features of the training data.AdaBoost combines multiple weak classifiers to form a strong classifier by iteratively adjusting the weights of incorrectly classified instances[30, 31]. The weight of each data point is updated on each iteration to increase the weight of incorrectly classified instances. Given weights w_i for each data point i, the update rule is:

$$w_{i,t+1} = w_{i,t} \times \exp\left(\alpha_t \times I(y_i \neq h_t(x_i))\right) \tag{2.6}$$

where $w_{i,t+1}$ is the updated weight for iteration $t+1$, α_t is the weight of the weak classifier h_t at iteration t, and $I(y_i \neq h_t(x_i))$ is an indicator function that is 1 if y_i is not equal to the prediction $h_t(x_i)$ and 0 otherwise[32]. The weight of each weak classifier h_t is determined based on its accuracy:

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \text{err}_t}{\text{err}_t}\right) \tag{2.7}$$

where err_t is the error rate of h_t [32].

Deep Neural Networks : Deep Neural Networks (DNNs) are powerful machine

learning models suitable for multiclass classification problems across high-dimensional data. Their architecture contains an input layer for raw data, hidden layers with ReLU activation to learn patterns, and an output layer using softmax for probability-based class predictions. A DNN consists of an input layer, multiple hidden layers, and an output layer[33]. Each layer comprises nodes or neurons, with the hidden layers using activation functions to introduce non-linearity, enabling the model to learn complex patterns[33]:

- Input Layer: Receives raw data. The dimensionality of the input layer corresponds to the number of features in the dataset. - Hidden Layers: Transform the input data into something the output layer can use. Typically utilize ReLU (Rectified Linear Unit) activation for intermediate layers due to its efficiency and simplicity[34, 35].

- Output Layer: Produces the prediction. For multiclass classification, it uses the softmax activation function, which converts logits to probabilities for each class.

The expression for the output at layer l in a neural network can be represented as:

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]} \tag{2.8}$$

where $Z^{[l]}$ is the output of layer l, $W^{[l]}$ and $b^{[l]}$ are the weight matrix and bias vector for layer l, and $A^{[l-1]}$ is the activation from the previous layer[36, 37, 38]. The activation $A^{[l]}$ for each layer is computed using an activation function f, such that:

$$A^{[l]} = f(Z^{[l]}) \tag{2.9}$$

For the ReLU activation function is defined as:

$$f(z) = \max(0, z) \tag{2.10}$$

The final output $A^{[L]}$ for the last layer L uses the softmax function, where each element is computed as:

$$\frac{e^{Z_i^{[L]}}}{\sum_k e^{Z_k^{[L]}}} \tag{2.11}$$

This converts the logits into probabilities for each class.

The loss function for multiclass classification is typically the categorical crossentropy, given by:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{2.12}$$

where M is the number of classes, y is a binary indicator of whether class label c is the correct classification for observation o, and p is the predicted probability that observation o is of class c [39].

- Integration with Side-Channel Analysis

  The integration of ML models with side-channel analysis comprises many fundamental methodologies:

  - Feature Extraction: This is an important step where significant features are recovered from the side-channel data (e.g., power traces, timing information). Features might include statistical metrics, frequency components, or particular patterns that are suggestive of malicious behavior.

  - Model Training: The extracted characteristics are utilized to train the ML model. This step comprises choosing a suitable algorithm, adjusting hyperparameters,

and utilizing training data that contains both benign and Trojan-inserted instances.

- Detection and Classification: Once trained, the model is used to categorize new samples as either benign or Trojan-infected. The efficacy of detection is highly dependent on the quality of feature extraction and the model's capacity to generalize from the training data[2, 9, 40].

- Effectiveness and Efficiency : The Random Forest Classifier and Deep Neural Networks showed better precision and durability. This model's performance is attributed to its ensemble learning technique, where numerous decision trees contribute to a final choice, lowering the danger of overfitting. This model has the ability to accommodate varied data sets and is adept at catching complicated, non-linear patterns. This makes it extremely dependable for identifying hardware Trojans, which typically display subtle and sophisticated signatures. Precision and recall are critical metrics for evaluating the performance of classification models. They are defined as follows:

  - Precision is the ratio of true positive predictions to the total number of positive predictions (both true positives and false positives)[41]:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.13}$$

  -Recall is the ratio of true positive predictions to the total number of actual positives (both true positives and false negatives):

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.14}$$

  Where:

  – TP (True Positives) is the number of correct positive predictions.

- FP (False Positives) is the number of incorrect positive predictions.
- FN (False Negatives) is the number of incorrect negative predictions [42].

-In terms of efficiency,Support Vector Machines and K-Nearest Neighbors have shown commendable performance. However, as the level of complexity of the dataset rises, both may experience scaling issues. Support Vector Machines, necessitating certain kernel choices, might become resource-intensive. Neural Networks, especially Deep Neural Networks, despite their excellent accuracy metrics, may suffer inefficiencies owing to their large processing demands and the requirement for huge training datasets, particularly in situations with resource limits[43, 42, 41].

- Comparative Analysis

  Recent work suggests a trend towards the use of Random Forest Classifier and deep Neural Network models for Trojan detection, ascribed to their greater capacity to identify meaningful patterns from complicated datasets. However, these models need enormous computing resources and big amounts of labeled data for training, which may not always be viable.

Integrating machine learning with side-channel analysis to detect hardware Trojans is is a prominent and a rapidly growing area of research. While Random Forest Classifier and deep learning models show great effectiveness. It is important to choose other models that achieves a balance between accuracy and computational efficiency. This is particularly important when there are limitations on resources. Choosing the right model requires careful analysis of the specific requirements of the work, taking into account factors like as the speed of detection, the amount of training data available, and computing limitations. Ongoing advancements in machine learning and side-channel methods continuously strengthen our defensive strategies.

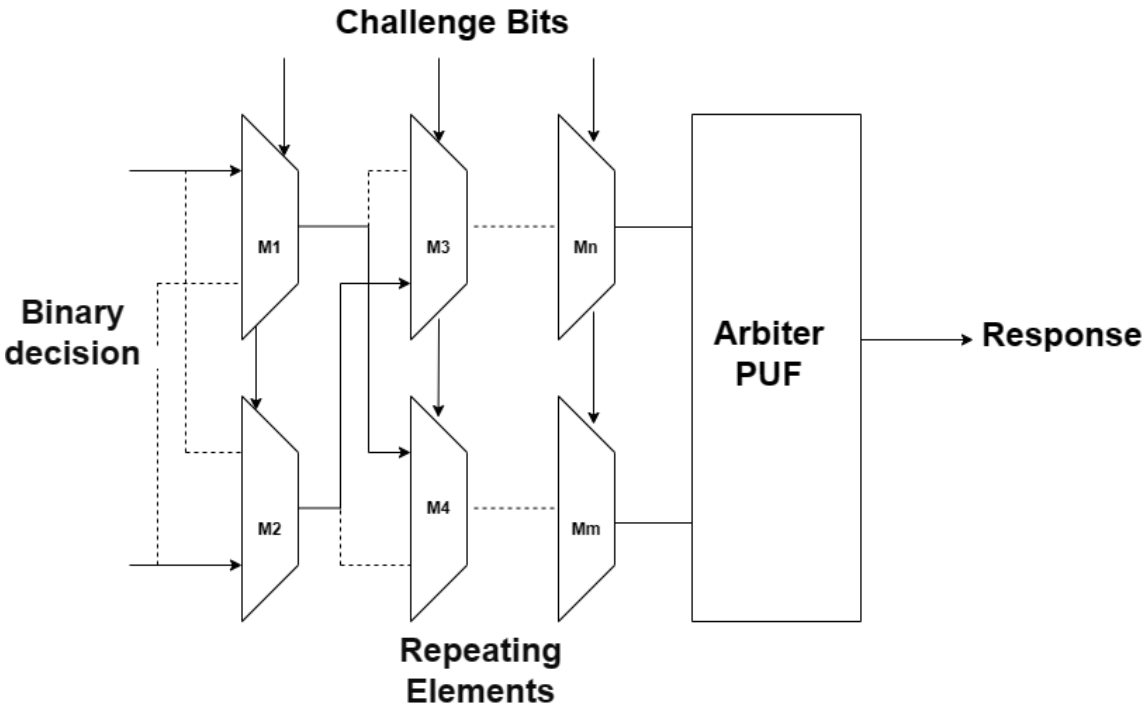

Figure 2.2: Arbiter PUF Structure

2.3 Physically Unclonable Functions: From Foundations to Frontiers

Physically Unclonable Functions (PUFs) use the inherent imperfections created in hardware during manufacture to provide unique IDs or cryptographic keys. These intrinsic variances are unpredictable and hence make PUFs an excellent tool for hardware authentication and security. Among the different forms of PUFs, Arbiter PUFs are remarkable for their simplicity and effectiveness[44].

- Principles of PUFs and Arbiter PUFs

  PUFs operate by the application of a challenge to a physical system and Assessing the response , where the reaction is strongly reliant on the physical properties of the system. The uniqueness of each PUF instance Complicates to copy or guess responses without physical access.

Arbiter PUFs especially employ a competitive situation between two identical signal channels that have been transformed differentially by manufacturing variations. At the end of each of the pathways, an arbiter (a basic flip-flop or latch) decides which signal arrived first as shown in Fig [2.2]. The input challenge configures the pathways in a manner that impacts the signals' travel times, and the arbiter's decision delivers the binary response. This method enables Arbiter PUFs to produce responses depending on the inherent physical characteristics of the device[45]. The operation of an Arbiter Physically Unclonable Function (PUF), which uses race conditions to generate responses, might be abstractly represented as follows:

Given two signal paths, path1 and path2, the difference in travel times due to manufacturing variations is denoted as Δt_{path1} and Δt_{path2}, respectively. The Arbiter PUF produces a binary response R based on these differences, which can be mathematically represented as:

$$R = A(\Delta t_{path1}, \Delta t_{path2}) \tag{2.15}$$

Here, R is the binary response determined by the arbiter A based on the difference in travel times (Δt) of signals along the two paths.

- Use of PUFs in Hardware Authentication

  PUFs are widely used for hardware authentication as they provide each device with a distinct"fingerprint" that can be validated against a predetermined value. This application is critical in situations like supply chain integrity, counterfeit detection, and secure key creation, when the authenticity of a device has to be certified without relying on readily replicated digital certificates or stored keys.

- Machine Learning Attacks on PUFs

Even while PUFs, such as Arbiter PUFs, provide security benefits, they are nevertheless susceptible to certain sorts of attacks. Machine learning (ML) attacks have emerged as a serious concern. In such attacks, an attacker utilizes ML techniques to develop a model of the PUF by evaluating recorded challenge-response pairs (CRPs). If these ML assaults are effective, they may predict the PUF's responses to new challenges, thus replicating the PUF without having physical access[46].

Composite or hybrid Physically Unclonable Functions (PUFs) aim to complicate direct modeling by combining multiple PUF responses. If PUF_1 and PUF_2 are two different PUFs, a composite response could be represented as:

$$R_{composite} = \text{Combine}\left(PUF_1(C_1), PUF_2(C_2)\right) \tag{2.16}$$

Where *Combine* is a function that merges the responses from PUF_1 and PUF_2 to a set of challenges C_1 and C_2, respectively.

- Literature Review on Machine Learning Attacks

  Numerous research in the literature have carefully evaluated different ML attacks on PUFs, exposing the susceptibility of even complicated PUF designs to modern ML models. These studies highlight the necessity for PUF designs that are robust against modeling attacks, Initiating a relentless pursuit of research to produce PUFs that can survive such challenges.

- Emerging Trends and Advancements in PUF Security

  Recent innovations in increasing PUF security emphasize the development of architectures and protocols which inherently offer greater defiance to ML attacks. Some of them include:

  - Composite and Hybrid PUFs: By merging several PUF structures or kinds,

these designs attempt to expand the complexity of the challenge-response behavior, making it difficult for ML algorithms to predict effectively.

- Error Correction and Helper Data Algorithms: These approaches increase the dependability of PUF responses while simultaneously obfuscating the actual nature of the PUF's physical properties, making it more difficult for attackers to create accurate models.

- PUF-based Cryptographic Protocols: Significant strides in this realm include incorporating PUFs into cryptographic protocols in a manner that uses their unique properties for increased security, such as key agreement procedures that necessitate the unpredictability of PUF answers.

Arbiter PUFs including other PUF present potential remedies for hardware authentication and security, benefiting from the the unique physical properties of devices. However, the potential of machine learning attacks has driven extensive research into more robust PUF designs and security techniques. As this realm unfolds, the emphasis remains on designing PUFs that balance the needs of security, dependability, and usability in an increasingly complex risk scenario.

2.4 Vulnerabilities and Defenses: A Machine Learning Perspective

Hardware authentication systems, crucial to fortifying the integrity and validity of devices across a multitude of applications, meet significant challenges from ML attacks. These attacks operate with ML algorithms to delve into the actions of authentication methods, particularly Physically Unclonable Functions (PUFs), allowing attackers to imitate or override security mechanisms.. Research in this field has fully outlined vulnerabilities and created a variety of defense mechanisms to prevent such attacks[46]. This blends conclusions from recent research findings, highlighting

adaptive and resilient approaches that have been suggested to increase the security of hardware authentication systems.

- Documented Vulnerabilities to Machine Learning Attacks

  Machine learning attacks focus on addressing the predictability and repetition of responses in hardware authentication systems. For example, through the inspection of a sufficient amount of CRPs from a PUF-based system, attackers might train models to clone the PUF's behavior, Aptly bypassing the authentication process. These vulnerabilities originate from:

  - Linear and Simple Relationships: Simpler to grasp relationships within the hardware's response creation cycle. - Insufficient Entropy: Predictable or insufficiently unpredictable responses that enable ML algorithms to rapidly learn the system's behavior.

  - Static Behavior: Unchanging reaction patterns throughout time, which do not adjust to potential risks.

- Adaptive and Resilient Defense Mechanisms

  Addressing these shortcomings, research has been oriented towards designing defensive mechanisms that are both adaptable and durable to ML attacks. These techniques include:

  1. Nonlinear PUF Designs: By introducing more elaborate , nonlinear components into PUF designs, researchers intend to make it substantially difficult for ML algorithms to accurately mirror the PUF's behavior. These designs generally include hybrid or composite PUF structures that integrate different PUF technologies to generate more random CRPs.

  2. Continuous Reconfiguration : Some methodologies encompass dynamically reconfiguring the PUF or authentication mechanism in response to Detected

or presumed attacks. This might include modifying Methods for generating challenge-response patterns or Periodically revising the system's internal framework to invalidate whatever model an attacker may have created.

3. Noise Introduction: Intentionally injecting noise into the response generation process may effectively enhance the challenge for ML models attempting to learn the system's behavior. This noise must be carefully regulated to prevent diminishing the system's dependability for authorized users.

4. Use of Helper Data Algorithms: Helper data algorithms may increase security by obfuscating the link between challenges and responses. When paired with error-correction approaches, they not only increase resistance to ML attacks but also solve concerns of response dependability owing to environmental fluctuations or device aging.

5. Advanced Cryptographic Techniques: Integrating hardware authentication methods with modern cryptographic approaches, such as zero-knowledge proofs, may give extra levels of protection. These solutions enable a device to establish its authenticity without providing the real data or patterns that may be utilized in an ML attack.

- Synthesis of Research Findings

  Previous research outcomes suggest that a multimodal strategy, comprising numerous adaptive and resilient methods, delivers the best possible defense against ML attacks on hardware authentication systems. There isn't a singular method that is infallible, and the efficiency of each protection mechanism might vary according to the distinctive hardware and application situation. Ongoing innovation in both attack and defense strategies is essential, highlighting the continuing competition in hardware security.

Incorporating these adaptive and resilient defenses signifies crucial stages forward for hardware authentication systems in increasing threat environment provided by machine learning attacks. As the complexity of both hardware systems and ML approaches grows, the requirement for proactive, adaptive security solutions becomes more crucial.

2.5 Theoretical Frameworks Informing Detection and Authentication

ML is often used in hardware security to primarily identify hardware Trojans and ensure the security of PUFs. This application is substantiated by several theoretical models and frameworks. These fundamental principles not only determine the development of ML algorithms but also provide the methodologies for using these technologies in hardware security circumstances. In this study, we explore the crucial theoretical principles and their implementation in the fields of Trojan detection and PUF protection.Machine learning encompasses a wide range of theoretical models.

1. Supervised Learning: This method relies on datasets that have been labeled to train the algorithm on how to make predictions or classifications. Supervised learning may be used in the field of hardware security to detect hardware Trojans. This is done by training models on circuits that are known to be infected with Trojans and circuits that are Trojan-free. By doing so, the algorithm can learn the distinguishing characteristics in between.

2. Unsupervised Learning: Unsupervised learning algorithms detect patterns or anomalies without the need for labeled data. This technique is particularly crucial for detecting distinct or previously encountered hardware Trojans because accurate fingerprints may not be pre-established.

3. Reinforcement Learning: Despite being less often used in the field of hardware

security, reinforcement learning involves the use of algorithms that learn optimal behaviors via a process of trial and error in order to maximize a reward. The idea of this may be studied for dynamic security systems that respond to evolving risks, such as adaptive PUF settings.

4. Semi-supervised and active learning models use both labeled and unlabeled data, which is uniquely valuable when complete labeled datasets are scarce or expensive to create. They may increase the efficiency and accuracy of Trojan detection algorithms by actively searching the most informative unlabeled instances for labeling.

Frameworks for ML Applications in Hardware Security

Feature Selection and Extraction: Prerequisite for data preprocessing in either case Trojan detection and PUF security, feature selection, and extraction processes assist in determining the foremost essential hardware signals (e.g., power consumption, timing information) to enhance learning efficiency and detection accuracy.

Anomaly Detection Frameworks: These frameworks serve as crucial tools for analyzing abnormalities in hardware behavior that might imply the presence of a Trojan or an attack on PUF integrity. Anomaly detection is particularly critical for unsupervised and semi-supervised learning approaches.

Adversarial Machine Learning: This theory highlights the concept of adversaries actively manipulating data to evade detection or change the learning process. It is especially crucial for developing robust ML models capable of recognizing complicated hardware Trojans and ensuring the security of PUFs against modeling attacks. Relevance to Detecting Trojans and Securing PUFs

Detecting Hardware Trojans: The use of supervised learning models, reinforced by feature selection and extraction approaches, enables the formulation of algorithms

that can effectively distinguish between clean and compromised hardware. Anomaly detection frameworks significantly enhance the ability to recognize new Trojans.

Securing PUFs: The integrity and unpredictability of PUF responses are critical for hardware security. ML models, particularly those focusing on anomaly detection and adversarial machine learning, may be deployed to examine PUF activities, identify possible weaknesses, and create defenses against cloning or modeling attacks.

Adversarial Resilience: The conceptual foundations of adversarial machine learning are essential for anticipating and mitigating attempts against ML-based security solutions. They encourage the development of more robust algorithms that can withstand attempts to imitate or upset the normal functioning of hardware components.

In conclusion, the theoretical models and frameworks of machine learning provide an extensive basis for overcoming the issues of hardware security. By applying these principles, researchers and experts may construct complex tools and techniques for recognizing hardware Trojans and increasing the security of PUFs, ensuring the integrity and trustworthiness of hardware devices in the midst of increasing threats.

Frameworks for ML Applications in Hardware Security

Feature Selection and Extraction: Prerequisite for data preprocessing in either case Trojan detection and PUF security, feature selection, and extraction processes assist in determining the foremost essential hardware signals (e.g., power consumption, timing information) to enhance learning efficiency and detection accuracy.

Anomaly Detection Frameworks: These frameworks serve as crucial tools for analyzing abnormalities in hardware behavior that might imply the presence of a Trojan or an attack on PUF integrity. Anomaly detection is particularly critical for unsupervised and semi-supervised learning approaches.

Adversarial Machine Learning: This theory highlights the concept of adversaries actively manipulating data to evade detection or change the learning process. It is especially crucial for developing robust ML models capable of recognizing complicated hardware Trojans and ensuring the security of PUFs against modeling attacks.

Relevance to Detecting Trojans and Securing PUFs Detecting Hardware Trojans: The use of supervised learning models, reinforced by feature selection and extraction approaches, enables the formulation of algorithms that can effectively distinguish between clean and compromised hardware. Anomaly detection frameworks significantly enhance the ability to recognize new Trojans.

Securing PUFs: The integrity and unpredictability of PUF responses are critical for hardware security. ML models, particularly those focusing on anomaly detection and adversarial machine learning, may be deployed to examine PUF activities, identify possible weaknesses, and create defenses against cloning or modeling attacks.

Adversarial Resilience: The conceptual foundations of adversarial machine learning are essential for anticipating and mitigating attempts against ML-based security solutions. They encourage the development of more robust algorithms that can withstand attempts to imitate or upset the normal functioning of hardware components.

In conclusion, the theoretical models and frameworks of machine learning provide an extensive basis for overcoming the issues of hardware security. By applying these principles, researchers and experts may construct complex tools and techniques for recognizing hardware Trojans and increasing the security of PUFs, ensuring the integrity and trustworthiness of hardware devices in the midst of emerging threats.

2.6 Related Work

This chapter reviews the literature surrounding hardware security, particularly focusing on advancements in hardware Trojan detection and PUF security through machine learning (ML) techniques. It integrates the study within the ongoing existing research, highlighting how this work extends and diverges from previous efforts.

- Historical Context and Evolution of Hardware Security The evolution of hardware security unfolds as a compelling story, advancing with technological developments advancements. Its annals are distinguished by a continual conflict with the development of security measures on one side and the creation of threats, notably the more complicated hardware Trojans, on the other.

- The Dawn of Hardware Security Concerns

  In the early days of computing, security-related concerns about hardware were very confined. The emphasis was focused on usefulness and performance, with security concerns being at most secondary to the security notion. However, as computer systems grew increasingly vital to both military and commercial uses, the significance of safeguarding hardware against manipulation and espionage started to acquire awareness.

- The Rise of Hardware Trojans

  The phrase "hardware Trojan" originated in the mid-2000s, marking a new age of risks when harmful functionalities may be incorporated into the hardware itself, typically during the manufacturing process. These Trojans were intended to stay dormant until activated, making them exceptionally difficult to detect without compromising the normal operation of the device[47].

- Evolution of Threats and Detection Techniques

- Late 2000s: Recognition of the risk posed by hardware Trojans sparked the first research initiatives focusing on detection techniques. Most of these early approaches were generally unsophisticated, depending on visual inspection or functional testing, which proved unsuccessful against sophisticated Trojans.

- 2010s: The complexity of hardware Trojans proliferated, with attackers leveraging the deep supply chain and globalization of semiconductor production. This decade witnessed the development of more complex detection methods, such as side-channel analysis, which evaluates power usage, electromagnetic emissions, or timing information to find abnormalities suggestive of a Trojan.

- Mid-2010s to Early 2020s: Researchers began analyzing a multitude of innovative approaches for identifying risks involved including use of machine learning algorithms to evaluate hardware designs for unexpected patterns. Techniques like logic testing, which entails subjecting the hardware to a series of test vectors and evaluating the outputs for differences, also gained popular.

- Recent Developments: The emphasis has turned towards the design of extensive frameworks that integrate various detection approaches, exploiting the capabilities of each to increase detection rates. There has been a rising focus on the design-for-security paradigm, where hardware is engineered from the bottom up to be robust against Trojans.

- Significant Milestones

  - 2008: The first systematic taxonomy of hardware Trojans gave a framework for understanding and analyzing these risks.

  - 2012: Introduction of new side-channel analytic approaches for distinguishing Trojans, representing a substantial improvement over existing approaches.

  - 2015: The advent of the first machine learning-based approaches for hardware Trojan detection, highlighting the significance of AI in fortifying cybersecurity.

- 2018: Implementation of hardware security implementations in retail merchandise, demonstrating a shift towards mainstream acceptance of the risks associated with hardware Trojans[48].

- Existing works on Differential Power Analysis, establishing the relevance of power side-channel signals in detecting covert hardware manipulations and the integration of power side-channel analysis with ML to enhance Trojan detection efficacy, marking a significant methodological shift pays vital role in the Power Side-Channel Analysis Techniques and Applications [49, 50].

- The application of ML in detecting hardware Trojans has transformed the landscape of hardware security. From the early use of Support Vector Machines in identifying Trojans , towards the recent efforts to explore the potential of deep learning techniques, highlighting the evolving sophistication of ML approaches in this domain bring a paradigm shift towards Machine Learning in Hardware Trojan Detection [51, 52].

- From Foundations to Frontiers Discussions on PUFs underscore their critical role in hardware authentication and the vulnerabilities exposed by machine learning-based attacks. a comprehensive analysis of PUF architectures and their susceptibilities, framing the ongoing challenge of securing PUFs against sophisticated threats provides significant contribution towards the work. The dual impact of ML in both compromising and defending hardware security is scrutinized. Previous work presents a pivotal examination of ML's potential to bypass traditional security measures, while subsequent studies propose innovative defense mechanisms to counteract these vulnerabilities [53, 54].

- Theoretical Frameworks Informing Detection and Authentication Exploring the theoretical underpinnings of ML applications in hardware security, Previous works on adversarial training, which offers valuable insights into developing ML

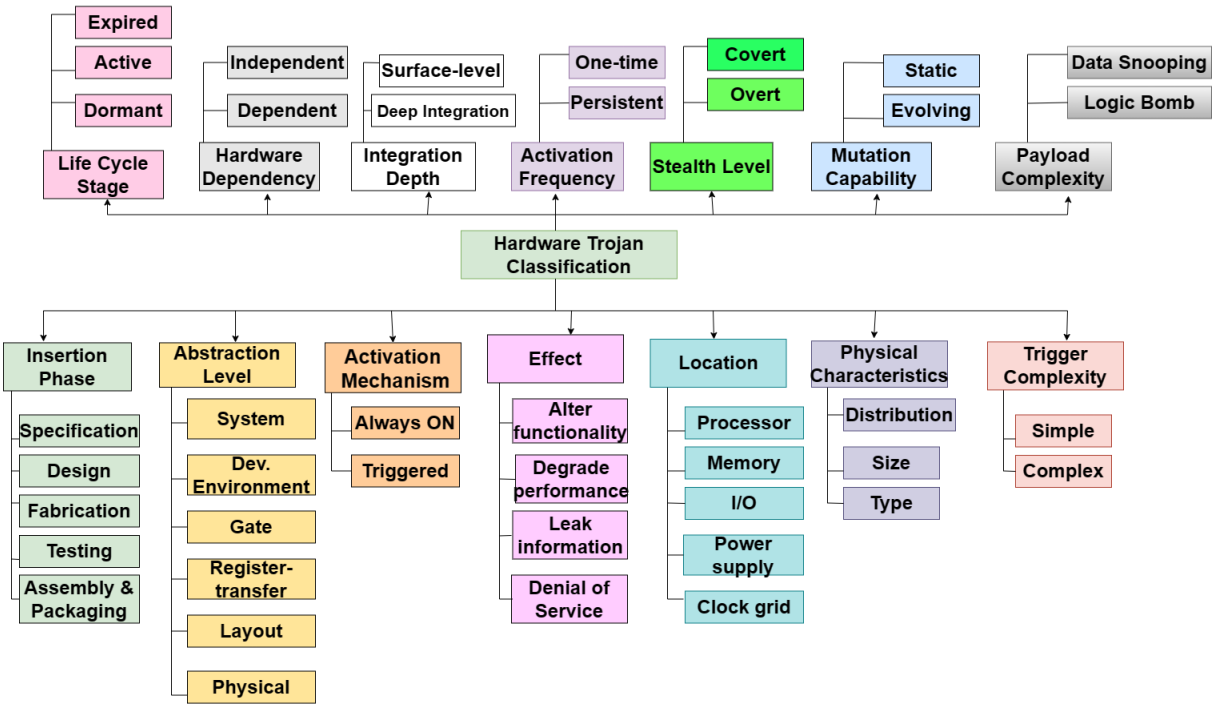

Figure 2.3: Hardware Trojan classification

models resilient to attack strategies. Such theoretical frameworks provide the basis for designing robust security solutions [55]. -Critique of Current Literature A critical review of the literature reveals gaps in research, particularly in the areas of dynamic security solutions and real-world applicability. The synthesis of findings marks the need for further investigation into adaptive ML models that can respond to emerging threats with agility.

The development of hardware security highlights the dynamic interaction between technological improvements and the changing nature of threats. With our growing dependency on electronic devices in every area of our lives, the significance of securing hardware against Trojans and other destructive agents is important. This growth from primitive security mechanisms to the sophisticated detection and prevention approaches illustrates the research and development community unwavering determination to protecting the security of our digital world.

2.7 Critique of Current Literature

While the use of machine learning (ML) in hardware security, notably in identifying hardware Trojans and fortifying physically unclonable functions (PUFs), has seen tremendous progress, there are significant drawbacks, contradictions, and places where existing research may fall short. Addressing these gaps is essential for developing research and functional applications in this particular field[15].

-Limitations of Current ML Approaches

1. Data Availability and Quality:A key restriction is the unavailability of publicly accessible, high-quality datasets for training and testing ML models, particularly datasets that precisely demonstrate real-world scenarios of hardware Trojans and PUF setups. This insufficiency impedes the progress and validation of strong ML models.

2. Generalization Capability: A multitude of ML models are challenged by overfitting, causing them to perform well on training data but badly on unknown data. This challenge is especially crucial in hardware security, as attackers continually adapt their techniques and models must adapt effectively to emerging threats.

3. Resource limits: Developing ML algorithms for real-time detection in hardware may be tricky owing to the computational and power limits of many devices, particularly embedded systems or IoT devices.

4. Adversarial attacks: The literature often neglects the susceptibility of ML models to adversarial attacks, where slight effective adjustments to the input may lead to incorrect model predictions. This vulnerability is an important issue for security applications.

-Discrepancies in research findings

1. Inconsistent assessment criteria: There is a lack of standardization in assessments and benchmarks amongst researchers, which makes it impossible to evaluate the effectiveness of various ML approaches for hardware Trojans or securing PUFs.

2. Objectives and Scope: The concept of what qualifies a "hardware Trojan" or a "secure PUF" could vary greatly among studies, resulting in conflicts in the goals and findings of various research efforts.

-Future improvements Areas :

1. Adversarial ML: Adversarial attacks are considered to be a major kind of concern. there is need for the improvement in the hardware security with defensive techniques.

2. Scalability and Efficiency: Additional efforts are very important to develop ML models that are not only effective in vulnerabilities but are also scalable and efficient enough to be used in resource-constrained conditions.

3.Real-World Applicability: Most of the previous research shows mathematical expressions without applying them on real world scenarios. 4.Interdisciplinary Approaches: It plays vital role as interdisciplinary approaches deals with the combination of cybersecurity, hardware design approaches and ML approaches to solve the Complex problems of hardware security. 5.Dynamic and Adaptive Systems: Upcoming research must focus on dynamic and adaptive ML models that can respond to new threats as static models deal with the problem of becoming obsolete rapidly.

In conclusion, application of ML for hardware security plays a vital role in addressing previous works limitations and filling gap in the previous studies relating with hardware security. To design more secure and robust hardware systems, there is

need of collaboration with various disciplines, improvement in getting the datasets, measures and focusing more on real world applications.

2.8 Research Gaps and Emerging Opportunities

The modern research framework for machine learning (ML) applications for hardware security, notably in the context of identifying hardware Trojans and improving the security of PUFs, displays numerous glaring gaps. These gaps not only illustrate the limits of current techniques but also promise fertile ground for additional inquiry and innovation. Addressing these deficiencies gives the area the area the opportunity to progress greatly, pushing the limits of what is currently attainable in hardware security.

- Gaps in Current Research

1. Adaptive and Dynamic Security Solutions: There is a gap in the design of ML models that can adapt effectively to new threats. Many current solutions are fixed in their design, restricting them from independently improving or expanding their knowledge base.

2. Comprehensive Benchmarks and Defined Datasets: The absence of defined datasets and benchmarks for assessing hardware security solutions is a serious gap. This makes it difficult to assess the success of various techniques and restricts the creation of generally applicable solutions.

3. Adversarial Attack Resistance: The susceptibility of ML models to adversarial attacks, particularly in a hardware security setting, is underexplored. There is a need for research focusing on improving ML models that may withstand or rapidly recover from attacks of this kind.

4. Integration of ML into Low-Power and Resource-Constrained Devices: Many ML models need large computing resources, which creates a hurdle for their integration into resource-constrained devices typical in hardware security applications.

5. Real-World Implementation Challenges: There is a gap between theoretical ML models and their actual implementation in real-world hardware systems. Issues such as environmental unpredictability, device wear and tear, and operational limits are commonly disregarded in research.

- Opportunities for further investigation and innovation

1. Developing Self-Learning Systems: Creating ML models that can learn and adapt in real-time to new threats has big potential. Such systems might autonomously update their knowledge base, making them particularly effective against developing hardware Trojans and other security threats.

2. Creation of Open-Access, High-Quality Datasets: There is a potential to produce and distribute comprehensive datasets that represent the complexity of real-world hardware security concerns. This would substantially assist in the training and testing of more robust ML models.

3. Advancements in Adversarial Machine Learning: Exploring new strategies for making ML models resistant to adversarial manipulation provides a road to more secure applications. This involves developing innovative training approaches, defense measures, and model designs.

4. Lightweight ML Models for Hardware Security: Implementing efficient, lightweight ML models that can function on low-power and resource-constrained devices might revolutionize the sector, making sophisticated security features available to a larger variety of devices.

5. Bridging Theory and Reality: Focusing research efforts on the actual application of ML models in hardware security, especially in the field testing and real-world case simulations, will help bridge the gap between theory and reality. This involves overcoming deployment issues, environmental adaptation, and long-term dependability.

By addressing these gaps, we have the potential to pave new paths in hardware security. Each of these domains presents distinct problems, but also offers the possibility of large benefits in terms of better security, dependability, and applicability of ML models in defending against hardware risks. Collaboration across disciplines, including machine learning, hardware engineering, and cybersecurity, will be key in moving these developments forward.

3 Proposed Methodology

3.1 Detecting Hardware Trojans

3.1.1 Experimental Setup

This section outlines Experimental procedures and design approaches to identify FPGAs utilizing advanced machine learning techniques. Considering the upward trend of complexity and relevance of protecting integrated circuits from Malicious modifications. Our study helps for the need for effective detection methodologies introduced for identifying hardware Trojans that may be injected at any step of the chip's life cycle.

- Controlled Environment Setup: The experiment was conducted Under controlled conditions to simulate real-world under which hardware Trojans might be triggered. Specifically, FPGAs were chosen as the platform. widespread use and vulnerability to Trojan attacks are the reason why we choose it. A set of unique hardware Trojans, identified using Trust Hub benchmarks. These were injected into the FPGA boards to analyze their properties and the effectiveness of the detection methods. This approach helps for a precise assessment of Trojan behavior. Under controlled conditions, ensure that the results are reliable and reproducible[56, 57].

- Quantitative approach: A quantitative approach was initiated to assess and analyze the side-channel signals. Power and electromagnetic (EM) emissions are considered to be very important. these range from emerging from the FPGA boards.This

method requires collecting a complete dataset of these signals. Various operational conditions, including different Trojan states (disabled, enabled, and triggered) and environmental characteristics (such as chip external temperature) are considered to be essential for it. Machine learning algorithms were applied to the data set to make a model capable of distinguishing between normal and abnormal signal patterns. this is indicative of Trojan activity.

- Structure of Experiments :The experimental format was developed to capture a broad spectrum of data points. this might be across many variables. These variables include physical factors (power consumption and EM radiation), Trojan benchmarks (12 different situations), Trojan-prone conditions (disabled, enabled, triggered), input vector configurations, and chip external temperature of 25°C . Each experiment involves collecting 10,000 time-series signals. these are under specified conditions, synced with AES encryption cycles to imitate real-world operating scenarios. This detailed data set collection was automated using an FPGA testbed. This contains a test control program, oscilloscope, FPGA board, and temperature controller. The automated setup ensured great precision and efficiency in data collection. This was necessary for creating a successful machine learning model for hardware Trojan detection.

- Rationale for the Approach: There are two motivations for selecting this experimental setup. Firstly, it helps overcome the limitations of previous approaches. such approaches typically rely on the existence of a 'golden chip' for comparison. also existing models provides less accuracy detect Trojans and sometimes remain inactive throughout testing phases. Secondly, it makes use of machine learning to evaluate data from the side channel. Presenting a viable route for finding irregularities of Trojan activity is an important aspect to be considered. This methodology improves detection capabilities. Moreover,it also adds to the development of dynamic models

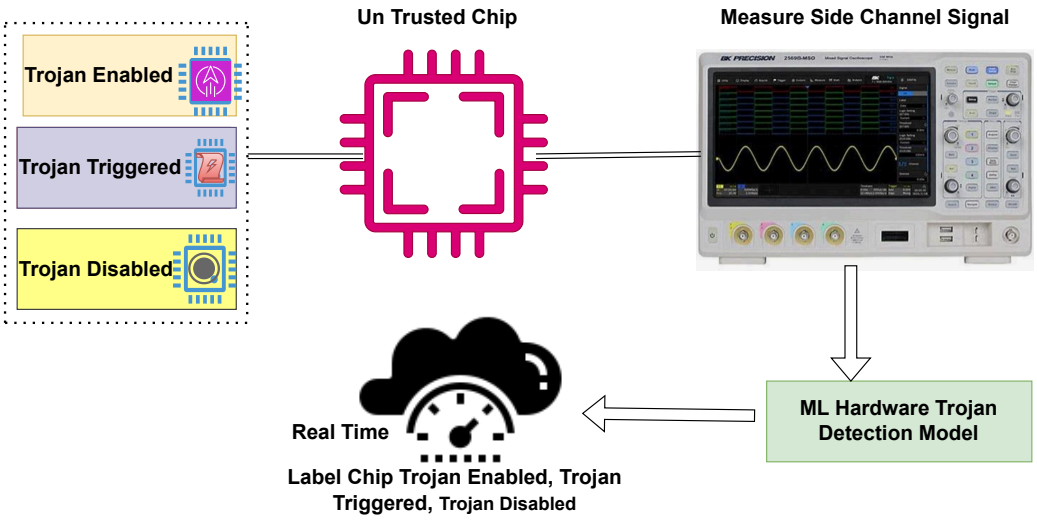

Figure 3.1: Proposed work for Hardware Trojan classification

that can adapt to new and developing Trojan threats over time.

The proposed work is more effective and reliable hardware Trojan detection tools. Further analysis is done following Integrating a controlled environment configuration with comprehensive quantitative analysis and machine learning algorithms. our goal is to secure a deeper insight into features of hardware Trojan.this helps in developing detection mechanisms. this gets Prepared to defend critical digital world against these threats.

3.1.2 Data Collection

This section presents an in-depth review of the approaches applied for collecting power side-channel signals from FPGA boards, essential for the identification of hardware Trojans. The collection method is important for accruing a solid dataset that allows the training and validation of our machine learning model

- Equipment and Measurement Techniques: The collection of side-channel signals was accomplished utilizing a complete setup designed to capture power usage and

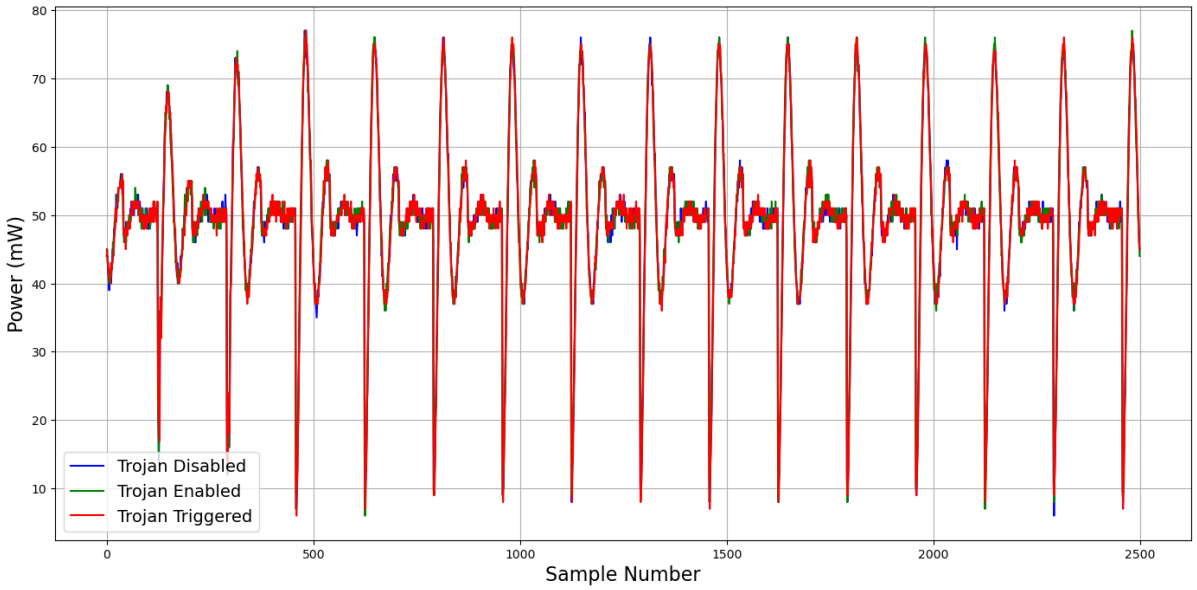

Figure 3.2: Power side-channel time-series signal for AES-2000 benchmark

electromagnetic emissions with great precision. The primary equipment includes:

- SAKURA-G FPGA boards: Selected for their reconfigurability and prevalence in research, permitting a direct mapping of our findings to real-world applications.

- Oscilloscope: Employed for recording high-resolution power usage patterns across time.

- Spectrum Analyzer: Used for measuring the electromagnetic emissions of the FPGA boards. Temperature Controller: To simulate various operational conditions and analyze their effects on the signals.

- Data collection frequency and duration: The approach to data collection was meticulously outlined to ascertain extensive coverage of potential operational states. Each FPGA board was tested under different conditions, with 10,000 time-series signals collected for each scenario. This extended data acquisition technique ensured a rich dataset, boosting the model's capacity to generalize across varied conditions.

-Pre-processing steps Prior to analysis, the obtained raw data underwent multiple pre-processing processes to enhance signal quality and relevance. These steps include: - Noise Reduction: Filtering techniques were developed to decrease surrounding and equipment-induced noise.

- Normalization: Ensure that signal amplitudes are within a comparable range to enable analysis.

Feature extraction: Identifying and extracting key properties from time series data to improve the performance of machine learning models.

- Selection and Variation of FPGA Boards : A diverse assortment of FPGA boards was selected to ensure that the conclusions of the study are universally applicable. The boards were chosen based on their employment in key applications, availability, and variance in architectural features. This variability allowed for the assessment of Trojans' impact across diverse hardware setups.

- Expert Involvement: Experts in hardware security and FPGA design were involved in the validation and annotation of the data set. The selection procedure for these was centered on their published work and contributions to the area of hardware security. Their knowledge was important in verifying the accuracy of the data annotation and in providing insights into details of Trojan behaviors not immediately apparent through automated analysis alone. through collection of data and pre-processing efforts, together with expert validation, underlie the resilience of our machine learning-based method to hardware Trojan identification. By encompassing a wide range of operational contexts and applying a methodical approach, this effort initiates a solid foundation for future advancements in assuring the security of integrated circuits against hostile modifications.

3.1.3 Data Analysis

The method of detecting patterns in power consumption to identify the presence of a Trojan comprises a detailed feature extraction step from the power consumption data, instead of straightforwardly deploying specific machine learning methods. The technique can be stated as follows, including features of machine learning methodology suited to the context provided by the code:

Feature extraction: Since there is only one feature, there is a need for more features that are related to obtaining a vast array of features from the power consumption data. These traits encompass different domains:Time-Domain Features: Including statistical measurements such as mean, median, standard deviation, skewness, and kurtosis, among others. These features help to capture the basic and detailed patterns within the power usage time series data.

Frequency-Domain Features: Using the Welch approach, features such as power spectral density, weighted mean frequency, and frequency variances are extracted. These features try to find distinctive signatures in the frequency components of the power consumption data, which may be suggestive of aberrant behavior.

Wavelet-Domain Features: The application of the wavelet transform enables the study of the power consumption signal at various scales, capturing both the frequency and the coordinates. Features obtained from wavelet coefficients provide insights into transient anomalies and non-linear patterns in the data.

Hypothetical Machine Learning Implementation: The next steps for anomaly detection may logically include:

Data Preprocessing: Before training machine learning models, it is important to preprocess the data. This may involve normalizing the characteristics. To ascertain

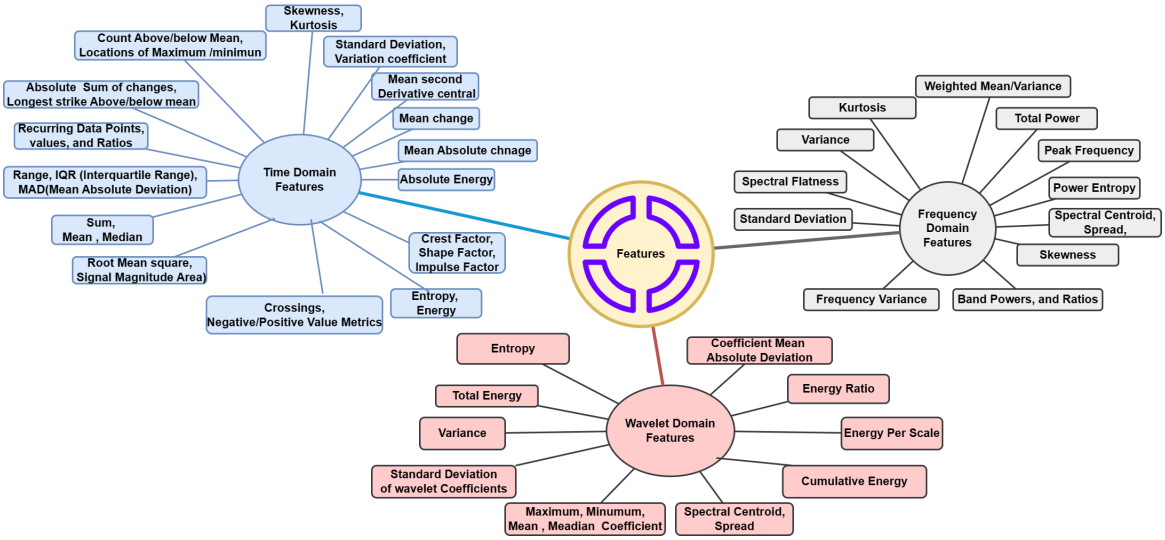

Figure 3.3: Various features used

that none of the factors disproportionately influences the model's predictions.

Model Selection and Training: While the code does not define machine learning models, based on the retrieved features, techniques such as random forest, support vector machines (SVM), or neural networks could be applied for anomaly detection.

Training Process: The selected model would be trained on a subset of the dataset, learning to distinguish between normal operating data and situations indicative of a Trojan based on the collected features.

Model testing and validation:Testing Process: The trained model is then evaluated on a distinct subset of the data (the test set) to evaluate its ability to accurately categorize unseen cases.

Cross-Validation: Employing techniques like k-fold cross-validation could further evaluate the model's performance across multiple subsets of the data.[58, 59, 60]

Performance Evaluation Metrics:The performance of the anomaly detection model can be evaluated using numerous metrics, including: Accuracy: the ratio of accurately

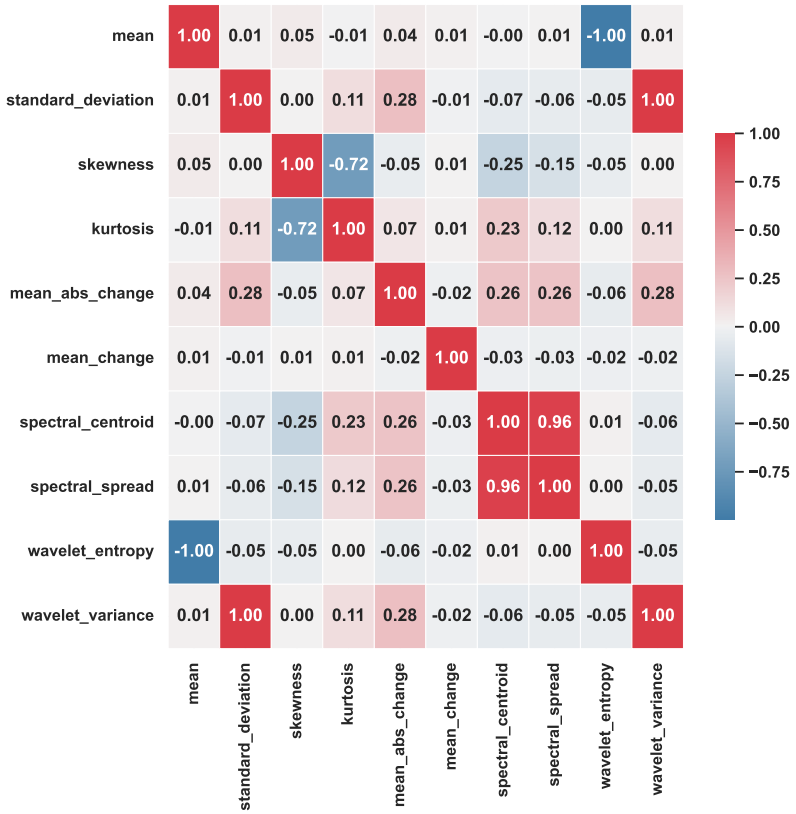

Figure 3.4: Correlation Matrix of features

predicted observations to the total observations, offering a basic measure of the model's performance.Precision, Recall (Sensitivity), and F1-Score: precision assesses the model's accuracy in forecasting anomalies; recall measures the model's capacity to detect all actual abnormalities; and the F1-Score provides a balance between precision and recall.

Confusion Matrix: Offers deep insights into true positives, false positives, true negatives, and false negatives, giving a sophisticated knowledge of model correctness. This comprises model training and testing, followed by performance evaluation utilizing multiple metrics to detect the existence of a Trojan successfully. [58]

3.1.4 Proposed Model

This model utilizes power consumption analysis to detect hardware Trojans, incorporating a wide range of features derived from time, frequency, and wavelet domains, as well as employing innovative dataset and preparation techniques. The following is a summary of the model. The model associate power consumption data obtained under various scenarios (Trojan deactivated, enabled, and triggered) from FPGA boards. This illustrates a comprehensive and systematic method for achieving and accelerating datasets that reflect different operational states of the hardware. The initial steps involve importing relevant libraries and suppressing warnings to speed up and expedite the data analysis process. Subsequently, data from numerous CSV files, which depict and represent different experimental situations, are combined, resulting in a comprehensive dataset for extracting features.

Then the process involves feature extraction, which are based on time, frequency, and wavelet domain features. We have extracted many features from these features.The feature collection sets a good starting point for applying multiple ML methods for Trojan detection. Our best model includes their interpretability and potential to manage with non-linear relationships. The performance of the model is evaluated using accuracy, precision, recall, and the F1 score. Additionally, confusion matrices show the model's ability to identify various states accurately.

This model, with feature extraction and use of machine learning algorithms is very important to detect hardware Trojans through power usage analysis. The extended feature set, including time, frequency, and wavelet domains, ensures the power consumption patterns,leading to high detection accuracy of hardware Trojans[61, 62]. The following are the best machine learning models that give highest accuracy :

- Random forest classifier: This proposed model plays an important role in

the prediction of classification of hardware Trojan's. In this proposed model, after pre-processing, scaling, and splitting the dataset and refining the input values were implemented. Then the random forest classifier is applied to the training data. RFC has different types of decision trees. each decision trees has a different vote. the class that has the maximum number of votes for the given input gets the best prediction of the model. The number of trees in the voting for a provided class can be considered as the confidence for its prediction.

To improve the Performance of the model, the above selected features were undergone through experiment that shows the presence of Trojan and can improve the detection capability.The improvement in selecting important features over time was directed by the feature significance scores of the model. updating hyperparameters plays an important role in this model.This includes the number of decison trees $n_{\text{estimators}}$, the maximum depth of trees max_depth, and the minimum samples required to split an internal node $min\_samples\_split$. Finally , the better accuracy is obtained by optimising the above parameters to get fit with the unique features of the hardware Trojan dataset.

- Deep Neural network : Another best model that provides an improvement in performance is the deep neural network (DNN). The framework of the model is designed using Keras with TensorFlow.The basic structure consists of various densely linked layers with rectified linear unit (ReLU) activation functions. This is followed by the layer known as softmax output layer for classification. The compilation of the model is done by Adam optimizer along with categorical cross entropy loss function. This gives the improvement in the best accuracy of the performance. likewise the other models, first of all the input features are standardized by using StandardScaler. then it follows the training process. after the completion of training on the selected dataset for 100 epochs is kept for further future use. Finally, the overall performance

of the classification is shown by using accuracy, the confusion matrix.

3.2 Securing Hardware Authentication

3.2.1 Dataset Preparation

The dataset is based on 64-stage Arbiter PUF.It consists of 12000 challenge response pairs (CRPs). where each CRP consists of a 64-bit challenge (columns 1-64) and a binary response (column 65), where the response indicates the PUF's response to the provided challenge.

For further analysis, the data set is divided into training and testing. the size of dataset were categorised into 2000,6000 and 10,000 samples. The samples were chosen randomly so that our findings are applicable anywhere. the remaining samples were considered for testing and prediction.the dataset consists of 1200 CRPs from a 64-stage Arbiter PUFs and are loaded into Pandas Data frame.[63].This initial step is very important for preprocessing and further analysis are based on loaded and formatted data.

For machine learning process, standardization is an essential step that deals with preprocessing. Describes the necessary features that vary in terms of size, range, and units. binary values (1 or 0) are challenge bits in our dataset. While they are uniform in scale, standardizing these features indicates that one and all have equal chances to affect the model's learning process.to obtain mean of 0 and standard deviation of 1, the StandardScaler from scikit-learn can be applied. This technique helps to mitigate imbalances towards features with higher variance and makes sure that the model's performance is not affected by the size of features in the dataset[40].

This involves creating new features from the existing features and are considered to be the creative part of machine learning model that improves performance of the

model.this study consists of novel approach of using XOR operations in between pairs of the original 64-bit challenge features. The XOR technique was chosen for its ability to take non-linear interactions between challenge bits. moreover, it helps in providing new features that represent whether the paired bits vary. This strategy is particularly informative for our dataset, as it simulates some features of how Arbiter PUFs analyze challenges, potentially knowing patterns that are not immediately seen with the original features alone.

The introduction of XOR-based features were expected to improve the dataset by giving additional information that could be useful for the ML model in capturing the complex, nonlinear interactions in the challenge-response behavior of PUFs. This feature engineering process not only serves to improve the prediction performance of the model but also offers a deeper insight into the nature of the data and the general operation of Arbiter PUFs.

With the help of standardization and feature engineering, a change was made to the dataset aimed at optimizing the machine learning model's capacity to learn and predict reliably. The preparation methods are crucial for addressing the high-dimensional, binary feature of the PUF dataset and establishing a reliable groundwork for the initiation of RFC and potentially other machine learning models. The rigorous preparation and preprocessing of the data reflect the methodological rigor of the project and the strategic approach to overcome the obstacles posed by predicting the behavior of Arbiter PUFs.

3.2.2 Data Collection

Generating CRPs from Arbiter PUFs and the subsequent usage of these datasets in machine learning models for both attacking and defensive scenarios constitute an important area of research in hardware security. This section deals with the

mechanisms involved in generating CRPs from Arbiter PUFs and how these datasets assist in the creation of resilient machine learning models aimed at understanding, attacking, and defending PUF-based systems.

- Generating CRPs from Arbiter PUFs Arbiter PUFs function on the basis of race situations in electrical circuits. They consist of a sequence of switchable pathways that lead to an arbitrator choosing the ultimate output based on which path signal arrives first.

1. Challenge generation: Challenges to an Arbiter PUF are generated by defining the states (0 or 1) of the switches in the PUF's path. In a 64-stage Arbiter PUF, a challenge is a 64-bit binary bit where each bit reflects the status of one stage or switch in the PUF.

2. Response generation: Following the introduction of a challenge, the PUF processes is the internal arrangement of switches and delays, resulting in a race condition that leads to a binary response. This response is highly sensitive to the particular physical features of the PUF, making it difficult to predict responses without physical access to the device.

3. Pairing and Storage: The challenge and its response are paired together as a CRP. Collecting a number of these CRPs produces a dataset that captures the behavior of the individual Arbiter PUF under varied challenge conditions.

- Use of CRP Datasets in Machine Learning Models : with the application of ML algorithms CRP datasets can predict the basic patterns of Arbiter PUFs for both attack and defense techniques in hardware security situations.

In attack part, the goal is to build a predictive model that can accurately predict the responses of unknown challenges. By training the samples of the above CRPs,ML

algorithms learn the unique challenge-response behavior of PUF.

In this context, success is achieved by creating a model that can operates as a duplicate copy of the PUF and resulting with the security framework of the PUF by allowing unauthorized access to protected areas by security measures

In defense deals with changes the predictability and robustness of PUF responses. Machine learning models are used to detect the patterns in the PUF's responses that could be exploited by attackers. This research assists in building more complicated PUF frameworks or in developing anomaly detection systems that can determine when a PUF's response pattern differs from the expected behavior, indicating a potential attack.

The training approach deals with dividing the collected CRP data set into training and testing samples. The training set is made to provide necessary instruction to the machine learning model about the PUF's challenge-response behavior, while the testing set is built to test the model's accuracy and unseen data.

1. Model Training: The training phase modifies the model parameters to minimize the differences between the expected and actual responses in the training set. This step might include techniques like cross-validation to defend against overfitting and make sure the model remains capable for various applications.

2. Model Evaluation: After training, the model's performance gets evaluated using the testing set. indicators, such as accuracy, precision, recall, and F1 score, evaluate the model's efficacy in predicting responses. High accuracy in this phase suggests a successful attack model or a robust defense mechanism, depending on the scenario.

By applying the above dataset and machine learning approaches, it can help to

improve the security and dependability of Arbiter PUFs. The interplay between the building of complex PUF designs and machine learning-based attacks or countermeasures defines the dynamic nature of the study in hardware security[10].

3.2.3 Data Analysis Methods

This section addresses the methodology for assessing the dataset, the logic behind the chosen approaches, and how these methods contribute to gaining insight into the dataset's underlying patterns and behaviors.

-Preprocessing and Feature Engineering: The analysis begins by loading the dataset using Pandas, an initiative that prepares the way for subsequent data manipulation and analysis. To identify the importance of feature scaling in machine learning, the 'StandardScaler' was applied to normalize the feature collection, removing the response variable. This normalization maintains a balanced representation of all features in the investigation, minimizing the possible influence of differing scales among the features.

A novel part of our research is the insertion of interaction features using logical XOR operations between pairs of original challenge bits. This feature engineering stage is inspired by the concept that interactions between specific bits may reveal complicated patterns that a linear model could overlook. By altering the dataset to include these interaction terms, the study tries to capture the non-linear correlations that might be essential in forecasting the PUF responses accurately[64].

-Training and evaluation of machine learning models: implementation of a machine learning model like a Random Forest Classifier, a choice inspired by its adaptability and robustness in processing complicated, high-dimensional data. The model was trained on scaled features, including the newly constructed XOR interaction characteristics,

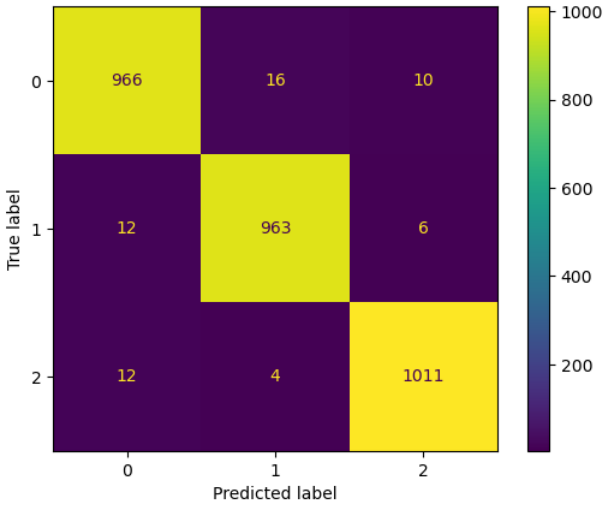

Figure 3.5: Confusion matrix for Random Forest Classifier

and tested for its accuracy in predicting responses to unseen challenges. The methodology includes separating the dataset into training and testing sets with changing ratios, reflecting a rigorous approach to understanding the model's performance across diverse training settings. 'GridSearchCV' for hyperparameter tuning embodies a thorough search for the ideal model configuration, employing cross-validation to boost the model's generalizability.

-Visualization and Descriptive Statistics: Visualization plays a significant role in our approach, with distribution plots of the response variable providing insights on the dataset's balance. Such representations help contextualize the model performance measures by emphasizing potential biases or unevenness in the data set. Moreover, the application of descriptive statistics delivers a quantitative overview of the data's features. This stage is critical for finding any outliers or anomalies that can influence the model's learning process and for ensuring that the dataset is well understood before stepping into predictive modeling.

The data analysis approaches discussed here, from preprocessing to feature engineering and the application of machine learning models, are aimed at carefully analyzing the

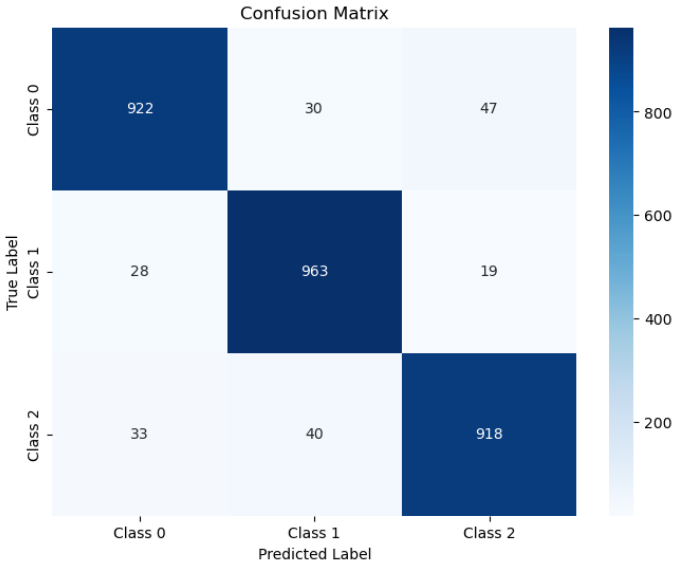

Figure 3.6: Confusion matrix for Deep Neural Network

dataset's patterns. The integration of visualization and statistical analysis significantly improves the comprehension of the dataset, ensuring a holistic approach to exposing the intricacies of Arbiter PUF behavior. Through these methodologies, the study not only intends to reliably forecast PUF reactions but also to contribute to a broader understanding of hardware security in the context of PUF technologies.

3.2.4 Proposed work

In the proposed work, this study focus on attack and defense mechanisms using machine learning models.we used various ML learning models for the prediction. RFC gives the best accuracy of all in the attack part.The attack component deals with creating interaction features through the logical XOR operation in between pairs of original features. This step is designed to complex and non-linear relationships that might enhance the model's predictive accuracy, especially in scenarios where interactions between features significantly impact the response variable.

The defense part starts with the conversion of all column names to strings. this is done to avoid compatibility issues. this helps in ensuring the model's resilience

against potential future warnings and errors.

Moreover, Gaussian noise is added to the response variable. this helps in simulating real-world where data may come with inherent noise.Preparing the model to handle such uncertainties are very important. This algorithm contains uniform noise into response data for robustness. It also performs a binary distribution evaluation across the first 64 columns to get an idea about imbalances.It also performs by calculating the mean of the relative differences in distribution and providing the overall balance of the dataset. It analyzes the counts of binary responses (1s and 0s) in a particular column to assess data balance.with the help of total counts, the absolute difference between the counts and calculating the difference in terms of percentage, it helps to understand the overall potential bias.

4 RESULTS AND DISCUSSION

The pursuit of robust cybersecurity protocols is connected to innovation and exploitation. Within this thesis, our analysis has explored the benefits of improved detection mechanisms for hardware Trojans and the reinforcement of hardware authentication mechanisms against machine learning-based attacks. Assessing research queries, the goal was to prove that advanced detection methods can mitigate hardware Trojan risks. Additionally, to ascertain that strengthening the defense of Arbiter PUFs would make them less vulnerable to predictive machine learning attacks, The following sections provide a detailed analysis of empirical data, focusing on power side-channel signals for the purpose of hardware Trojan identification, as well as data on the interaction between machine learning and Arbiter PUFs.

The study of the hardware trojan from the comprehensive dataset of power side-channel signal analysis provides a good classification and advances in identifying Trojans. The graphical illustration below shows the raw data and its processed form, respectively, indicating fluctuations in power consumption that could indicate Trojan activity. Additionally, using machine learning methods and analyzing the comparative performance of each model Descriptive statistics offer a brief summary, with the mean power consumption for Trojan-infected devices being much greater than their clean counterparts, with a highlighted standard deviation that highlights the deceptive qualities of these Trojans. Additionally, the range of power fluctuation in infected circuits extends above the expected norms, which further corroborates our detection assumption.

Interpreting the data as part of hardware Trojan detection reveals significant findings. The results clearly demonstrate that our proposed machine learning models excel at identifying subtle variances in power signatures indicative of a Trojan's presence. The performance metrics described below show a graphical representation that reveals a high accuracy rate in Trojan classification, surpassing the benchmarks set by traditional methods. Notably, random forest classification and deep neural networks demonstrated remarkable precision, thus confirming the potential of these models to significantly enhance hardware security measures. These findings not only support our hypotheses but also lay the groundwork for more nuanced and advanced detection techniques that may preempt the ever-evolving threats posed by hardware Trojans.

Machine Learning Model	**Precision**	**Recall**	**F1 Score**	**Accuracy**
KNN	58%	57%	57%	57%
Logistic Regression	43%	44%	43%	44%
Random Forest Classifier	97%	97%	97%	97%
Gradient Boosting Classifier	46%	47%	46%	47%
Ada Boost	42%	42%	42%	42%
SVM	53%	53%	53%	53%
Deep Neural Network	90%	90%	90%	90%

Table 4.1: AES-T400 Performance Metrics for Machine Learning Models at 25°C

Machine Learning Model	**Precision**	**Recall**	**F1 Score**	**Accuracy**
KNN	81%	81%	81%	81%
Logistic Regression	76%	75%	75%	75%
Random Forest Classifier	99%	99%	99%	99%
Gradient Boosting Classifier	76%	76%	76%	75%
Ada Boost	72%	71%	71%	71%
SVM	76%	75%	75%	75%
Deep Neural Network	95%	95%	95%	95%

Table 4.2: AES-T500 Performance Metrics for Machine Learning Models at 25°C

Our analysis begins with a thorough examination of the challenge-response pairs (CRPs) data. Each CRP represents a unique interaction with the 64-stage arbiter PUF, a critical component in hardware security. We visualized the distribution

Machine Learning Model	Precision	Recall	F1 Score	Accuracy
KNN	60%	59%	59%	59%
Logistic Regression	54%	54%	53%	54%
Random Forest Classifier	98%	98%	98%	98%
Gradient Boosting Classifier	55%	55%	55%	55%
Ada Boost	52%	51%	51%	51%
SVM	62%	62%	62%	62%
Deep Neural Network	95%	95%	95%	95%

Table 4.3: AES-T600 Performance Metrics for Machine Learning Models at 25°C

Machine Learning Model	Precision	Recall	F1 Score	Accuracy
KNN	82%	82%	82%	82%
Logistic Regression	73%	73%	73%	73%
Random Forest Classifier	99%	99%	99%	99%
Gradient Boosting Classifier	75%	75%	75%	75%
Ada Boost	67%	67%	65%	67%
SVM	73%	73%	73%	73%
Deep Neural Network	82%	82%	82%	82%

Table 4.4: AES-T700 Performance Metrics for Machine Learning Models at 25°C

Machine Learning Model	Precision	Recall	F1 Score	Accuracy
KNN	100%	100%	100%	100%
Logistic Regression	100%	100%	100%	100%
Random Forest Classifier	100%	100%	100%	100%
Gradient Boosting Classifier	100%	100%	100%	100%
Ada Boost	100%	100%	100%	100%
SVM	100%	100%	100%	100%
Deep Neural Network	100%	100%	100%	100%

Table 4.5: AES-T800 Performance Metrics for Machine Learning Models at 25°C

Machine Learning Model	Precision	Recall	F1 Score	Accuracy
KNN	66%	66%	66%	66%
Logistic Regression	59%	60%	59%	60%
Random Forest Classifier	98%	98%	98%	98%
Gradient Boosting Classifier	59%	60%	59%	60%
Ada Boost	56%	57%	56%	57%
SVM	63%	64%	63%	64%
Deep Neural Network	95%	95%	95%	95%

Table 4.6: AES-T1000 Performance Metrics for Machine Learning Models at 25°C

Machine Learning Model	Precision	Recall	F1 Score	Accuracy
KNN	59%	58%	58%	58%
Logistic Regression	52%	53%	53%	53%
Random Forest Classifier	98%	98%	98%	98%
Gradient Boosting Classifier	52%	53%	52%	53%
Ada Boost	50%	51%	50%	51%
SVM	57%	57%	57%	57%
Deep Neural Network	93%	93%	93%	93%

Table 4.7: AES-T1100 Performance Metrics for Machine Learning Models at 25°C

Machine Learning Model	Precision	Recall	F1 Score	Accuracy
KNN	83%	84%	83%	83%
Logistic Regression	90%	90%	90%	90%
Random Forest Classifier	99%	99%	99%	99%
Gradient Boosting Classifier	89%	89%	89%	89%
Ada Boost	84%	80%	79%	80%
SVM	90%	90%	90%	90%
Deep Neural Network	99%	99%	99%	99%

Table 4.8: AES-T1300 Performance Metrics for Machine Learning Models at 25°C

Machine Learning Model	Precision	Recall	F1 Score	Accuracy
KNN	59%	58%	58%	58%
Logistic Regression	45%	46%	45%	46%
Random Forest Classifier	98%	98%	98%	98%
Gradient Boosting Classifier	50%	50%	50%	50%
Ada Boost	43%	43%	43%	43%
SVM	57%	57%	57%	57%
Deep Neural Network	93%	93%	93%	93%

Table 4.9: AES-T1400 Performance Metrics for Machine Learning Models at 25°C

Machine Learning Model	Precision	Recall	F1 Score	Accuracy
KNN	59%	57%	58%	57%
Logistic Regression	41%	42%	41%	42%
Random Forest Classifier	98%	98%	98%	98%
Gradient Boosting Classifier	44%	44%	43%	44%
Ada Boost	40%	40%	40%	40%
SVM	51%	51%	51%	51%
Deep Neural Network	90%	90%	90%	90%

Table 4.10: AES-T1600 Performance Metrics for Machine Learning Models at 25°C

Machine Learning Model	Precision	Recall	F1 Score	Accuracy
KNN	58%	57%	57%	57%
Logistic Regression	48%	48%	48%	48%
Random Forest Classifier	98%	98%	98%	98%
Gradient Boosting Classifier	50%	50%	50%	50%
Ada Boost	46%	46%	46%	46%
SVM	57%	57%	57%	57%
Deep Neural Network	93%	93%	93%	93%

Table 4.11: AES-T1800 Performance Metrics for Machine Learning Models at 25°C

Machine Learning Model	Precision	Recall	F1 Score	Accuracy
KNN	59%	58%	58%	58%
Logistic Regression	48%	48%	48%	48%
Random Forest Classifier	97%	97%	97%	97%
Gradient Boosting Classifier	52%	52%	52%	52%
Ada Boost	48%	47%	47%	47%
SVM	60%	60%	60%	60%
Deep Neural Network	93%	93%	93%	93%

Table 4.12: AES-T2000 Performance Metrics for Machine Learning Models at 25C

Table 4.13: Comparison with Previous Work

Model	Previous Work	Deep Neural Network	RFC
AES-T400	89.77%	90%	97%
AES-T500	98%	95%	99%
AES-T600	93.98%	95%	98%
AES-T700	100%	82%	99%
AES-T800	100%	100%	100%
AES-T1000	71.35%	95%	98%
AES-T1100	77.61%	93%	99%
AES-T1300	100%	99%	99%
AES-T1400	85.39%	93%	98%
AES-T1600	75.90%	90%	98%
AES-T1800	93.45%	93%	98%
AES-T2000	92.98%	93%	97%
Average	**89.86%**	**93.16%**	**98.33%**

and variety of the CRPs using heatmaps and dimensionality reduction techniques, providing an intuitive understanding of the dataset's complexity. We employed various machine learning models to assess their effectiveness in both attacking and securing the Arbiter PUFs. The models included Logistic Regression, K-Nearest

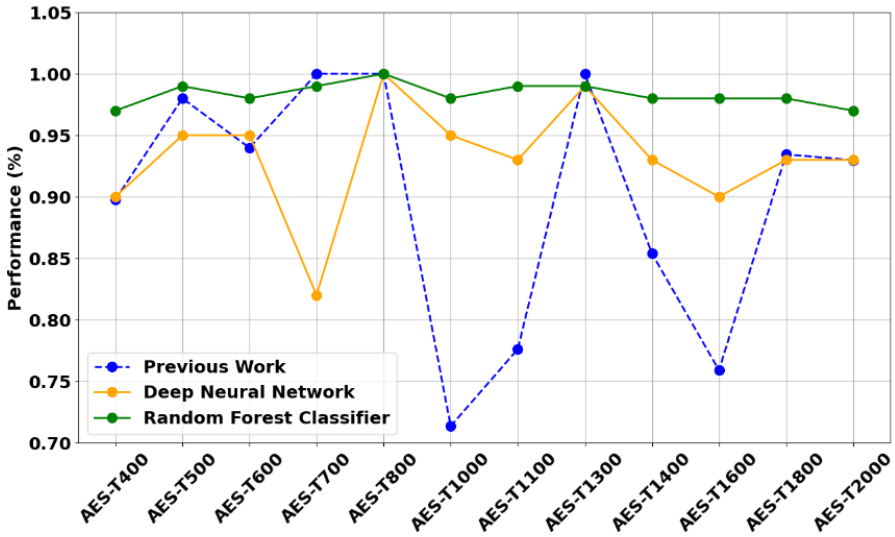

Figure 4.1: Comparison with previous work

Neighbors, Naive Bayes, Gradient Boosting, and notably, the Random Forest Classifier. The performance of these models was illustrated through detailed tables and bar charts, comparing the accuracy scores across different training set sizes.

The effectiveness of these strategies was graphically depicted, allowing us to identify which models performed best in compromising the PUFs and which were most effective in defending against attacks. It was observed that the accuracy of machine learning models fluctuated with the size of the training set, revealing the nuanced relationship between training data volume and model performance.

Comprehensive tables provided a clear comparison of performance metrics. For instance, the Random Forest Classifier illustrated a notable increase in effectiveness in the defense scenario with smaller training sets, an insight that could influence how we approach the training phase of security models.

The analysis revealed that certain machine learning models could predict PUF responses with higher accuracy, thereby posing a threat to the PUF's security. For example, the Random Forest Classifier achieved high success rates, suggesting that

PUF designs must be resilient against sophisticated ensemble methods.

Conversely, the defense strategies employed by the same models, especially with smaller training sets, indicated a promising direction for safeguarding PUFs. These strategies managed to reduce the effectiveness of simulated attacks, enhancing the PUF's robustness. Our findings contribute to the ongoing discourse on PUF vulnerabilities and defenses. They underline the necessity for adaptive and dynamic PUF designs capable of withstanding machine learning-based attacks. We also considered the current literature on PUF security, ensuring our research aligns with and contributes to the established body of knowledge. The project unearthed several pivotal insights, such as the inverse relationship between training set size and defense accuracy in certain contexts. These discoveries were substantiated by appropriate statistical analyses, strengthening the credibility of our conclusions. PUFs present a novel approach to hardware authentication, their susceptibility to machine learning attacks necessitates continuous refinement of their design and the defense mechanisms protecting them. Our study sets the stage for future research into more resilient PUF architectures and sophisticated defense algorithms.

The synthesis of results from the two pivotal projects reveals a complex narrative of hardware security in the modern landscape. On one hand, the project on hardware Trojans unveiled a crucial trend: the Random Forest Classifier and Deep Neural Network models emerged as powerful tools, surpassing traditional methods in detecting minute fluctuations in power signatures. On the other hand, the Arbiter PUFs project highlighted the susceptibility of seemingly robust security measures to sophisticated machine learning attacks, with the same Random Forest model proving to be a double-edged sword—adept at both attacking and defending.

The commonality across both projects lies in the profound impact of machine learning. It serves as a versatile weapon, capable of both undermining and reinforcing

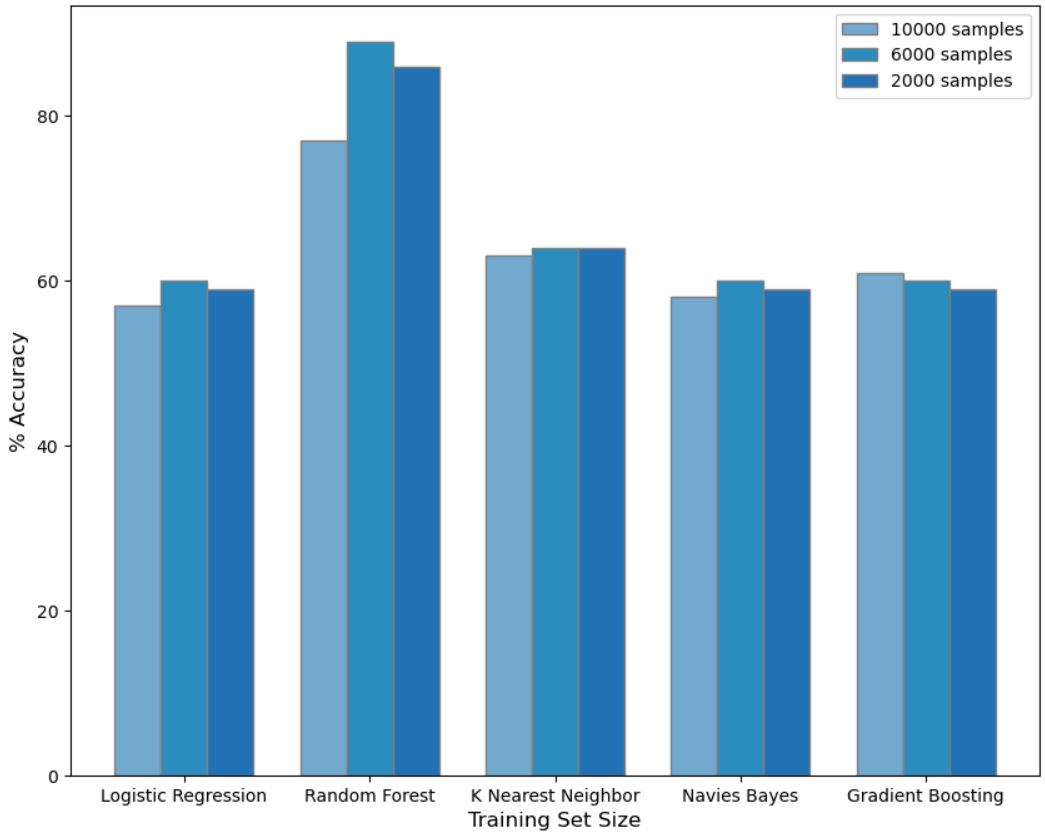

Figure 4.2: Attack on arbiter PUF

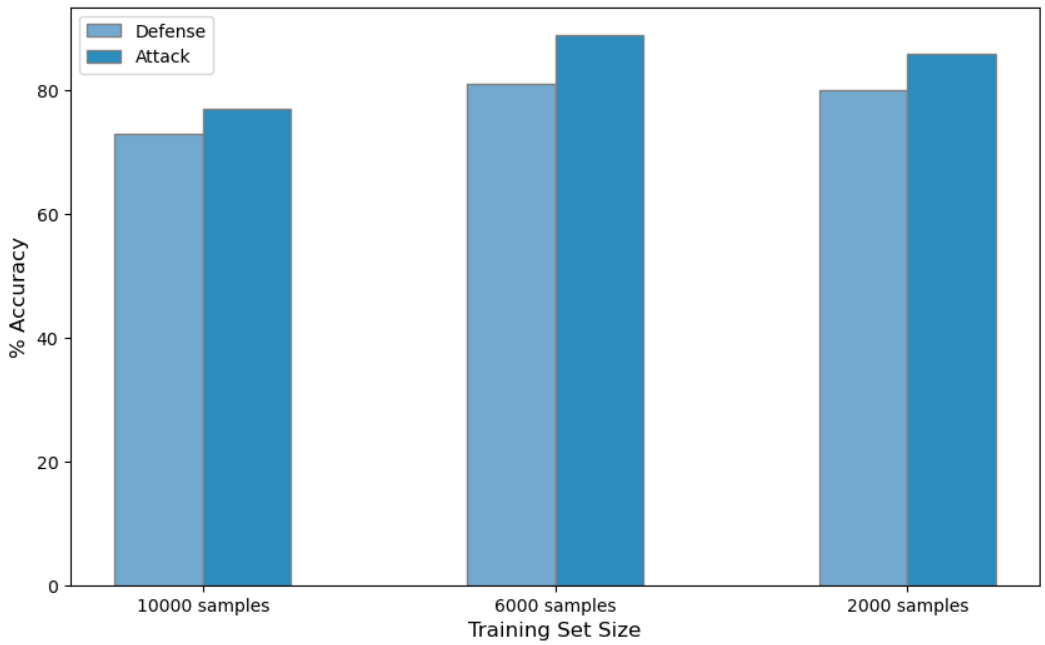

Figure 4.3: Comparison of attack and Defence on arbiter PUF

hardware security. The results suggest an intricate balance in the broader field of hardware security, where the tools developed for protection can also be harnessed for breach, thereby driving a constant need for innovation.

Integrating these results with the theoretical framework discussed earlier, it becomes evident that while the technology holds immense potential for security advancements, it also requires cautious and deliberate application to prevent and counteract threats. The practical implications resonate with the literature that advocates for dynamic and adaptive security measures to stay ahead in this perpetual game of innovation versus exploitation.

exploration into hardware Trojans demonstrated that advanced machine learning techniques could detect subtle signs of compromise with high accuracy, validating our hypothesis and contributing to a more secure digital environment. Meanwhile, examination of Arbiter PUFs shed light on the vulnerability of these devices to machine learning-based attacks, urging a re-evaluation of their role in cryptographic security.

These findings not only affirm the hypotheses posited at the thesis's outset but also paint a multifaceted picture of machine learning's role in hardware security—a tool for both fortification and infiltration. As we transition to the subsequent chapter, we will contextualize these findings within the existing body of knowledge. The implications for the field are substantial, as they call for a continuous cycle of reinforcement and evaluation of cybersecurity measures in the face of advanced technological threats and capabilities.

5 CONCLUSION AND FUTURE WORK

This study has addressed the emergent challenges within the realm of hardware security by several orders of magnitude.It also has focused on Improvement in the detection of hardware Trojans and the vulnerabilities of Arbiter PUFs to machine learning attacks. RFC and Deep neural networks show better results for the identification of hardware Trojan. Meanwhile, these models additionally present an unusual risk to the security of Arbiter PUFs, since they have the capacity to decrypt and replicate PUF responses.

The results show the importance of maintaining accuracy while implementing ML techniques in the realm of hardware security. Although these techniques provide a significant enhancement in the identification and mitigation of security, it is important to understand the possibility of their exploitative practices. The inherent duality of these technologies that serves as both a protective barrier and an effective tool in the realm of cybersecurity has been clearly demonstrated. Based on our findings from this study, it is suggested that future research efforts should focus on various important domains to advance the domain of hardware security.

-Advanced PUF frameworks: Researchers in the future should focus on the design of complex PUF frameworks which are resistant to machine learning-based modeling. The study of innovative PUF designs might involve the use of random processes that are more unpredictable resulting to be more secure.

-Dynamic and Adaptive Security Protocols: The creation of dynamic security

measures are able to react constantly to the changing nature of attacks. Machine learning models that continuously evolve and modify their parameters in real-time might prove to be more effective in identifying complex and adaptive hardware Trojans.

-Techniques Against ML Attacks: Efforts should be given towards development of improved defenses that can protect from machine learning attacks. This could involve the application of adversarial machine learning to generate PUF responses that are more unpredictable and more challenging to replicate.

-Legal and Ethical Guidelines: With the development of ML applications in the field of cybersecurity, there is also a need for robust legal and ethical guidelines to control the use and misuse of these advanced tools.

-Cross-Disciplinary Collaborations: Addressing the gap amongst hardware design, cybersecurity, and artificial intelligence through cross-disciplinary collaborations might encourage the development of new security solutions that take advantage of each domain.

To address these important domains, the field could evolve towards the manufacturing of hardware parts and structures which are not just secure by design but also resistant against the continuously evolving wide range of cyber-attacks. Through such focused and innovative studies, we might aim to establish a digital community that is safe, reliable, and trustworthy.

6 Appendix

6.1 Source code

This section deals with the necessary codes used in the study of hardware Trojan detection and authentication using ML assisted models.the dataset used for the hardware trojan detection is taken from the source: https://ieee-dataport.org/open-access/hardware-trojan-power-em-side-channel-dataset [65, 56].

6.2 Codes for the Design of ML models

Selection of feature plays vital role in the overall processing of the dataset.the more the features,the better results we can get.The following code deals with the feature selection for the overall process.

6.2.1 Feature Engineering

```
import pywt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew, kurtosis
from scipy.signal import welch
from scipy.signal import welch, find_peaks,
```

```
peak_prominences, peak_widths
from scipy.stats import entropy
import scipy
from scipy import stats
import warnings

# Suppress all warnings
warnings.filterwarnings("ignore")
```

6.2.2 Imbalanced Data Handling

```
def calculate_features(array, sampling_rate=1):
    # Time-Domain Features
    time_features = {
        'sum_values': np.sum(array),
        'abs_energy': np.sum(np.abs(array)),
        'mean_abs_change': np.mean(np.abs(np.diff(array))),
        'mean_change': np.mean(np.diff(array)),
        'mean_second_derivative_central':
        np.mean(np.diff(np.diff(array))),
        'median': np.median(array),
        'mean': np.mean(array),
        'standard_deviation': np.std(array),
        'variation_coefficient': np.std(array) / np.mean(array),
        'skewness': skew(array),
        'kurtosis': kurtosis(array),
```

```
'absolute_sum_of_changes':
np.sum(np.abs(np.diff(array))),
'longest_strike_below_mean':
max(np.where(array < np.mean(array))[0],
default=0),
'longest_strike_above_mean':
max(np.where(array > np.mean(array))[0],
default=0),
'count_above_mean': np.sum(array > np.mean(array)),
'count_below_mean': np.sum(array < np.mean(array)),
'last_location_of_maximum':
len(array) - np.argmax(array[::-1]) - 1,
'first_location_of_maximum': np.argmax(array),
'last_location_of_minimum':
len(array) - np.argmin(array[::-1]) - 1,
'first_location_of_minimum': np.argmin(array),
'percentage_of_reoccurring_datapoints_to_all_datapoints':
np.sum(array[1:] == array[:-1]) / len(array),
'percentage_of_reoccurring_values_to_all_values':
np.sum(array[1:] == array[:-1]) / len(array),
'sum_of_reoccurring_values':
np.sum(array[:-1][array[:-1] == array[1:]]),
'sum_of_reoccurring_data_points':
np.sum(array[1:] == array[:-1]),
'ratio_value_number_to_time_series_length':
np.sum(array != 0) / len(array),
'maximum': np.max(array),
```

```
'minimum': np.min(array),
'Range': np.max(array) - np.min(array),
'Interquartile Range (IQR)':
np.percentile(array, 75) - np.percentile(array, 25),
'Mean Absolute Deviation (MAD)':
np.mean(np.abs(array - np.mean(array))),
'Coefficient of Variation':
np.std(array) / np.mean(array),
'Root Mean Square':
np.sqrt(np.mean(np.square(array))),
'Signal Magnitude Area':
np.sum(np.abs(array) / len(array)),

'Sum of Squares': np.sum(np.square(array)),
'Entropy': entropy(array),
'Energy': np.sum(np.square(array)) / len(array),
'Root Mean Square of Successive Differences':
np.sqrt(np.mean(np.square(np.diff(array)))),
'Maximum Absolute Value': np.max(np.abs(array)),
'Minimum Absolute Value': np.min(np.abs(array)),
'Mean of Absolute Values': np.mean(np.abs(array)),
'Variance of Absolute Values': np.var(np.abs(array)),
'Standard Deviation of Absolute Values':
np.std(np.abs(array)),
```

```
'Number of Crossings (above mean)':
np.sum((array[:-1] < np.mean(array))
& (array[1:] >= np.mean(array))),
'Number of Crossings (below mean)': np.sum((array[:-1] >
np.mean(array)) & (array[1:] <= np.mean(array))),
'Mean of Positive Values': np.mean(array[array > 0]),
'Mean of Negative Values': np.mean(array[array < 0]),
'Number of Positive Values': np.sum(array > 0),
'Number of Negative Values': np.sum(array < 0),
'Percentage of Positive Values': np.mean(array > 0),
'Percentage of Negative Values': np.mean(array < 0),
'Sum of Positive Values': np.sum(array[array > 0]),
'Sum of Negative Values': np.sum(array[array < 0]),
'Ratio of Positive to Negative Sums':
np.sum(array[array > 0]) / np.sum(array[array < 0]),
'Crest Factor': np.max(np.abs(array)) /
np.sqrt(np.mean(np.square(array))),
'Shape Factor': np.sqrt(np.mean(np.square(array))) /
np.mean(np.abs(array)),
'Impulse Factor':
np.max(np.abs(array)) /
np.mean(np.abs(array)),
'Autocorrelation':
np.correlate(array,
array, mode='full')[len(array)-1],
'Zero Crossing Rate': np.mean(np.diff(array > 0)),
'Mean Square': np.mean(np.square(array)),
```

```
'Geometric Mean': scipy.stats.gmean(array),
'Harmonic Mean': 1 / np.mean(1 /
(np.abs(array) + 1e-9)),
'Trimmed Mean': scipy.stats.trim_mean(array,
proportiontocut=0.1),
'Sum of Exponential Values': np.sum(np.exp(array)),
'Exponential Mean': np.mean(np.exp(array)),
'Mean of Logarithmic Values':
np.mean(np.log(array))+ 1e-10,
'Sum of Logarithmic Values':
np.sum(np.log(array))+ 1e-10,
'Covariance': np.cov(array),
'Correlation Coefficient': np.corrcoef(array),
'Maximum Slope': np.max(np.diff(array)),
'Minimum Slope': np.min(np.diff(array)),
'Mean Slope': np.mean(np.diff(array)),
'Variance of Slope': np.var(np.diff(array)),
'Standard Deviation of Slope':
np.std(np.diff(array)),
'Sum of Absolute Differences':
np.sum(np.abs(np.diff(array))),
'Entropy of Differences':
entropy(np.diff(array)),
'Sum of Squares of Differences':
np.sum(np.square(np.diff(array))),
'Zero-crossing Density':
np.sum(np.diff(array > 0)) /len(array),
```

```
'Mean of Positive Differences':
np.mean(np.diff(array)[np.diff(array) > 0]),
'Mean of Negative Differences':
np.mean(np.diff(array)[np.diff(array) < 0]),
'Ratio of Mean Positive to Negative Differences':
np.mean(np.diff(array)[np.diff(array) > 0]) /
np.mean(np.diff(array)[np.diff(array) < 0]),
'Sum of Positive Differences': np.sum(np.diff(array)
[np.diff(array) > 0]),
'Sum of Negative Differences': np.sum(np.diff(array)
[np.diff(array) < 0]),
'Standard Deviation of Positive Differences':
np.std(np.diff(array)[np.diff(array) > 0]),
'Standard Deviation of Negative Differences':
np.std(np.diff(array)[np.diff(array) < 0]),
'Variance of Positive Differences': np.var(np.diff(array
[np.diff(array) > 0]),
'Variance of Negative Differences': np.var(np.diff(array)
[np.diff(array) < 0]),
'Peak-to-Mean Distance': np.max(array) - np.mean(array),
'Mean-to-Valley Distance': np.mean(array) - np.min(array),
'Normalized Total Variation':
np.sum(np.abs(np.diff(array))) / np.sum(array),
'Median Absolute Deviation (MAD)':
np.median(np.abs(array - np.median(array))),
'Waveform Length': np.sum(np.abs(np.diff(array))),
'Signal-to-Noise Ratio (dB)':
```

```
10 * np.log10(np.mean(array ** 2) / np.var(array)),
'Cumulative Sum': np.cumsum(array)[-1],
'Area under the Curve': np.trapz(array),
'Root Sum Square': np.sqrt(np.sum(array ** 2)),
'Ratio of Unique Values': len(np.unique(array)) / len(array),
'Mean of the Gradient': np.mean(np.gradient(array)),
'Variance of the Gradient': np.var(np.gradient(array)),
'Standard Deviation of the Gradient': np.std(np.gradient(array)),
'Entropy of Gradient': entropy(np.gradient(array)),
'First Derivative Root Mean Square':
np.sqrt(np.mean(np.square(np.gradient(array)))),
'Second Derivative Root Mean Square':
np.sqrt(np.mean(np.square(np.gradient(np.gradient(array))))),
'Cumulative Energy': np.cumsum(np.square(array))[-1],
'Energy of the First Derivative':
np.sum(np.square(np.gradient(array))),
'Energy of the Second Derivative':
np.sum(np.square(np.gradient(np.gradient(array)))),
'Zero-crossing Rate of First Derivative':
np.mean(np.diff(np.gradient(array) > 0)),
'Zero-crossing Rate of Second Derivative':
np.mean(np.diff(np.gradient(np.gradient(array)) > 0)),
'Mean Absolute Deviation of First Derivative':
np.mean(np.abs(np.gradient(array))),
'Mean Absolute Deviation of Second Derivative':
np.mean(np.abs(np.gradient(np.gradient(array)))),
'Range of First Derivative': np.max(np.gradient(array)) -
```

```
        np.min(np.gradient(array)),
        'Range of Second Derivative':
        np.max(np.gradient(np.gradient(array))) -
        np.min(np.gradient(np.gradient(array))),
        'Signal Magnitude Vector': np.sqrt(np.sum(array ** 2)),
        'Jerk (Third Derivative) Mean':
        np.mean(np.gradient(np.gradient(np.gradient(array)))),
        'Jerk (Third Derivative) Standard Deviation':
        np.std(np.gradient(np.gradient(np.gradient(array)))),
         'Jerk (Third Derivative) Energy':
         np.sum(np.square(np.gradient(np.gradient(np.gradient(array))))),
    freq, psd = welch(array, fs=sampling_rate,
    nperseg=min(len(array), 256))
    weighted_mean = np.sum(freq * psd) / np.sum(psd)
    weighted_variance = np.sum(psd *
    (freq - weighted_mean) ** 2) / np.sum(psd)

frequency_features = {
        'Weighted Mean': weighted_mean,
        'Weighted Variance': weighted_variance,
        'Sum of Squares': np.sum(np.square(array)),
        'Entropy': entropy(array),
        'Energy': np.sum(np.square(array)) / len(array),
        'Root Mean Square of Successive Differences':
        np.sqrt(np.mean(np.square(np.diff(array)))),
        'Maximum Absolute Value': np.max(np.abs(array)),
        'Minimum Absolute Value': np.min(np.abs(array)),
```

```
'Mean of Absolute Values': np.mean(np.abs(array)),
'Variance of Absolute Values': np.var(np.abs(array)),
'Standard Deviation of Absolute Values':
np.std(np.abs(array)),
'Number of Crossings (above mean)': np.sum((array[:-1] <
np.mean(array)) & (array[1:] >= np.mean(array))),
'Number of Crossings (below mean)': np.sum((array[:-1] >
np.mean(array)) & (array[1:] <= np.mean(array))),
'Mean of Positive Values': np.mean(array[array > 0]),
'Mean of Negative Values': np.mean(array[array < 0]),
'Number of Positive Values': np.sum(array > 0),
'Number of Negative Values': np.sum(array < 0),
'Percentage of Positive Values': np.mean(array > 0),
'Percentage of Negative Values': np.mean(array < 0),
'Sum of Positive Values': np.sum(array[array > 0]),
'Sum of Negative Values': np.sum(array[array < 0]),
'Ratio of Positive to Negative Sums': np.sum(array[array > 0]) /
np.sum(array[array < 0]),
'Crest Factor': np.max(np.abs(array)) /
np.sqrt(np.mean(np.square(array))),
'Shape Factor': np.sqrt(np.mean(np.square(array))) /
np.mean(np.abs(array)),
'Impulse Factor': np.max(np.abs(array)) /
np.mean(np.abs(array)),
'Autocorrelation': np.correlate(array, array, mode='full')
[len(array)-1],
'Zero Crossing Rate': np.mean(np.diff(array > 0)),
```

```
    'Mean Square': np.mean(np.square(array)),
    'Geometric Mean': scipy.stats.gmean(array),
    'Harmonic Mean': scipy.stats.hmean(array[array > 0]),
    'Trimmed Mean': scipy.stats.trim_mean(array,
    proportiontocut=0.1),
    'Weighted Average': np.average(array),  # Note: 'weights'
    parameter is missing, you might want to provide it
    'Sum of Exponential Values': np.sum(np.exp(array)),
    'Exponential Mean': np.mean(np.exp(array)),
    'Mean of Logarithmic Values': np.mean(np.log(array)),
    'Sum of Logarithmic Values': np.sum(np.log(array)),
    'Covariance': np.cov(array),
    'Correlation Coefficient': np.corrcoef(array),
}
features = {}
features['total_power'] = np.trapz(psd, freq)
features['peak_frequency'] = freq[np.argmax(psd)]
features['peak_power'] = np.max(psd)
features['power_entropy'] =
-np.sum(psd * np.log2(psd + 1e-12))
features['mean_frequency'] =
np.sum(freq * psd) / np.sum(psd)
features['median_frequency'] =
np.median(freq[np.where(psd > np.median(psd))])
features['spectral_centroid'] = np.sum(freq * psd) / np.sum(psd)
features['spectral_spread'] = np.sqrt(np.sum(((freq -
features['spectral_centroid']) ** 2) * psd) / np.sum(psd))
```

```
features['spectral_skewness'] = np.sum(((freq -
features['spectral_centroid']) ** 3) * psd) /
(features['spectral_spread'] ** 3 * np.sum(psd))
features['spectral_kurtosis'] = (np.sum(((freq -
features['spectral_centroid']) ** 4) * psd) /
(features['spectral_spread'] ** 4 * np.sum(psd))) - 3
features['band_power_delta'] =
np.trapz(psd[(freq >= 0.5) & (freq < 4)],
freq[(freq >= 0.5) & (freq < 4)])
features['band_power_theta'] = np.trapz(psd[(freq >= 4)
& (freq < 8)], freq[(freq >= 4) & (freq < 8)])
features['band_power_alpha'] = np.trapz(psd[(freq >= 8)
& (freq < 12)], freq[(freq >= 8) & (freq < 12)])
features['band_power_beta'] = np.trapz(psd[(freq >= 12)
& (freq < 30)], freq[(freq >= 12) & (freq < 30)])
features['band_power_gamma'] = np.trapz(psd[(freq >= 30)],
freq[(freq >= 30)])
features['power_ratio_delta'] = features['band_power_delta'] /
features['total_power']
features['power_ratio_theta'] = features['band_power_theta'] /
features['total_power']
features['power_ratio_alpha'] = features['band_power_alpha'] /
features['total_power']
features['power_ratio_beta'] = features['band_power_beta'] /
features['total_power']
features['power_ratio_gamma'] = features['band_power_gamma'] /
features['total_power']
```

```
# Calculate weighted variance
features['frequency_variance'] =
np.sum(psd * (freq - weighted_mean) ** 2) / np.sum(psd)
features['frequency_standard_deviation'] =
np.sqrt(features['frequency_variance'])
features['frequency_range'] = np.max(freq) - np.min(freq)
features['spectral_flatness'] =
np.exp(np.mean(np.log(psd + 1e-12))) / np.mean(psd)
features['spectral_slope'] =
np.polyfit(freq, np.log(psd + 1e-12), 1)[0]
features['spectral_decrease'] =
np.sum((psd[1:] - psd[:-1]) / freq[1:])
features['spectral_roll_off_85'] = freq[np.where(np.cumsum(psd)
>= 0.85 * np.sum(psd))[0][0]]
features['spectral_roll_off_90'] =
freq[np.where(np.cumsum(psd) >= 0.90 * np.sum(psd))[0][0]]
features['spectral_roll_off_95'] =
freq[np.where(np.cumsum(psd) >= 0.95 * np.sum(psd))[0][0]]
features['spectral_crest'] = np.max(psd) /
np.mean(psd)
features['zero_crossing_rate'] =
np.mean(np.abs(np.diff(np.sign(psd))))
features['energy'] = np.sum(psd ** 2)
features['power_spectrum_inertia'] =
np.sum((freq ** 2) * psd) / np.sum(psd)
features['max_autocorrelation'] =
```

```
np.max(np.correlate(psd, psd, 'full'))
features['autocorrelation_peak'] =
np.argmax(np.correlate(psd, psd, 'full'))
features['spectral_entropy'] =
-np.sum(psd / np.sum(psd) * np.log2(psd / np.sum(psd) + 1e-12))
features['spectral_energy_distribution'] =
np.sum(psd ** 2) / (np.sum(psd) ** 2)
features['spectral_flux'] = np.sqrt(np.sum(np.diff(psd) ** 2))
features['spectral_rolloff'] =
freq[np.where(np.cumsum(psd) >= 0.5 * np.sum(psd))[0][0]]
 features['spectral_variation'] = np.var(psd) /
 (np.mean(psd) ** 2)
features['normalized_spectral_entropy'] =
features['spectral_entropy'] / np.log2(len(psd))
features['frequency_mean_absolute_deviation'] =
np.mean(np.abs(freq - features['mean_frequency']))
features['spectral_edge_frequency_95'] =
freq[np.where(np.cumsum(psd) >= 0.95 * np.sum(psd))[0][0]]
features['spectral_edge_frequency_90'] =
freq[np.where(np.cumsum(psd) >= 0.90 * np.sum(psd))[0][0]]
features['spectral_edge_frequency_80'] =
freq[np.where(np.cumsum(psd) >= 0.80 * np.sum(psd))[0][0]]
# Calculate skewness and kurtosis of the power spectrum
features['power_spectrum_skewness'] =
stats.skew(psd)
features['power_spectrum_kurtosis'] =
stats.kurtosis(psd)
```

```
# Calculate peak to average power ratio
features['peak_to_average_power_ratio'] =
np.max(psd) / np.mean(psd)
features['peak_to_average_power_ratio'] =
np.max(psd) / np.mean(psd)
features['spectral_contrast'] = np.max(psd) / np.min(psd)
features['frequency_mode'] =
freq[np.argmax(np.bincount(np.digitize(freq,
np.arange(np.min(freq), np.max(freq), 0.1))))]
features['frequency_dispersion'] =
np.sqrt(weighted_variance)
features['spectral_skewness_normalized'] =
features['spectral_skewness'] /
(features['frequency_standard_deviation'] ** 3)
features['spectral_kurtosis_normalized'] =
features['spectral_kurtosis'] /
(features['frequency_standard_deviation'] ** 4)
features['spectral_smoothness'] =
np.sum(np.diff(np.diff(psd)) ** 2)
features['spectral_slope_2'] =
np.polyfit(freq, psd, 1)[0]
features['spectral_deviations'] =
np.sqrt(np.mean((psd - np.mean(psd)) ** 2))
features['spectral_rolloff_25'] =
freq[np.where(np.cumsum(psd) >= 0.25 * np.sum(psd))[0][0]]
features['spectral_rolloff_75'] =
freq[np.where(np.cumsum(psd) >= 0.75 * np.sum(psd))[0][0]]
```

```
features['power_spectrum_asymmetry'] = np.sum((freq <
features['mean_frequency']) * psd) / np.sum((freq >=
features['mean_frequency']) * psd)
features['cumulative_spectral_power_80'] =
np.sum(psd[np.cumsum(psd) <= 0.80 * np.sum(psd)])
features['cumulative_spectral_power_20'] =
np.sum(psd[np.cumsum(psd) <= 0.20 * np.sum(psd)])
features['spectral_bandwidth_2'] = np.sqrt(np.sum((freq -
features['mean_frequency']) ** 2 * psd)) / np.sum(psd)
features['spectral_bandwidth_3'] = np.power(np.sum((freq -
features['mean_frequency']) ** 3 * psd), 1/3) / np.sum(psd)
features['spectral_bandwidth_4'] = np.power(np.sum((freq -
features['mean_frequency']) ** 4 * psd), 1/4) / np.sum(psd)
features['spectral_complexity'] =
np.sum(psd > np.mean(psd))
features['spectral_phase'] = np.mean(np.angle(np.fft.fft(psd)))
features['spectral_impulse'] = np.max(psd) /
np.sqrt(np.mean(psd ** 2))
features['spectral_entropy_weighted'] =
-np.sum(psd * np.log2(psd + 1e-12) / np.sum(psd))
features['spectral_entropy_log'] =
-np.sum(np.log2(psd + 1e-12) * psd) / np.sum(psd)
features['frequency_quartile_1'] =
np.quantile(freq, 0.25, interpolation='midpoint')
features['frequency_quartile_3'] =
np.quantile(freq, 0.75, interpolation='midpoint')
features['interquartile_range'] =
```

```
features['frequency_quartile_3'] - features['frequency_quartile_1']
features['power_spectrum_density_peak'] =
np.max(psd / (np.sum(psd) * (freq[1] - freq[0])))
features['power_spectrum_density_mean'] =
np.mean(psd / (np.sum(psd) * (freq[1] - freq[0])))
features['power_spectrum_density_std'] =
np.std(psd / (np.sum(psd) * (freq[1] - freq[0])))
features['spectral_harmonicity'] =
np.sum(psd * np.cos(2 * np.pi * freq * np.argmax(psd))) / np.sum(psd)
features['spectral_inharmonicity'] =
np.abs(np.sum(psd * (np.sin(2 * np.pi * freq
* np.argmax(psd))))) / np.sum(psd)
# Calculate the 10% trimmed mean
features['frequency_trimmed_mean_10'] =
stats.trim_mean(freq, 0.1)
features['frequency_trimmed_mean_20'] = stats.trim_mean(freq, 0.2)
# Calculate the mean and maximum of the peak prominences
features['frequency_mean_absolute_difference'] =
np.mean(np.abs(freq - np.mean(freq)))
features['spectral_centroid_variance'] = np.var(psd * freq) /
(np.var(psd) * np.var(freq))
features['spectral_rise_time'] = freq[np.argmax(psd)] -
freq[np.argmin(psd)]
features['spectral_fall_time'] = freq[np.argmin(psd)] -
freq[np.argmax(psd)]
features['spectral_peak_to_mean_distance'] =
features['peak_frequency'] - features['mean_frequency']
```

```
features['spectral_mean_to_median_distance'] =
features['mean_frequency'] - features['median_frequency']
features['spectral_peak_count_above_mean'] = np.sum(psd >
np.mean(psd))
features['spectral_peak_count_below_mean'] = np.sum(psd
< np.mean(psd))
features['spectral_moments_mean'] =
np.mean([np.sum(freq ** i * psd) for i in range(1, 5)])
features['spectral_moments_variance'] =
np.var([np.sum(freq ** i * psd) for i in range(1, 5)])
spectral_moments = [np.sum(freq ** i * psd) for i in range(1, 5)]
features['spectral_moments_skewness'] = stats.skew(spectral_moments)
# Calculate spectral moments kurtosis
features['spectral_moments_kurtosis'] =
stats.kurtosis(spectral_moments)
features['spectral_slope_log'] =
np.polyfit(np.log(freq + 1e-12), np.log(psd + 1e-12), 1)[0]
features['spectral_slope_inverse'] =
np.polyfit(1 / (freq + 1e-12), psd, 1)[0]
# Wavelet-Domain Features
wavelet = 'db1'
level = 5
coeffs = pywt.wavedec(array, wavelet, level=level)
wavelet_coeffs = np.concatenate([c.flatten() for c in coeffs])
# Calculate the total energy for normalization
total_energy = np.sum([np.sum(np.abs(coeff) ** 2) for coeff in coeffs])
# Calculate wavelet total energy and entropy
```

```
features['wavelet_total_energy'] =
np.sum([np.sum(np.abs(sub_array) ** 2) for sub_array in coeffs])
features['wavelet_entropy'] =
-np.sum([np.sum((np.abs(sub_array) ** 2)
* np.log2(np.abs(sub_array) ** 2 + 1e-12))
for sub_array in coeffs])
# Calculate weighted variance
features['wavelet_variance'] = sum([len(sub_array) /
len(wavelet_coeffs) * np.var(sub_array) for sub_array in coeffs])
features['wavelet_standard_deviation'] =
np.sqrt(features['wavelet_variance'])
# Calculate various statistics on flattened coefficients
flattened_coeffs = np.concatenate
([coeff.flatten() for coeff in coeffs])
features['wavelet_maximum_coefficient'] = np.max(flattened_coeffs)
features['wavelet_minimum_coefficient'] = np.min(flattened_coeffs)
features['wavelet_mean_coefficient'] = np.mean(flattened_coeffs)
features['wavelet_median_coefficient'] = np.median(flattened_coeffs)
features['wavelet_coefficient_variance'] =
np.mean([np.var(coeff) for coeff in coeffs])
features['wavelet_coefficient_standard_deviation'] =
np.mean([np.sqrt(np.var(coeff)) for coeff in coeffs])
features['wavelet_energy_distribution'] =
np.mean([np.sum(np.abs(coeff) ** 2) /
total_energy for coeff in coeffs])
# Calculate skewness and kurtosis
features['wavelet_skewness'] = stats.skew(flattened_coeffs)
```

```
features['wavelet_kurtosis'] = stats.kurtosis(flattened_coeffs)
# Calculate peak to average power
features['wavelet_peak_to_average_power'] =
np.max(np.abs(flattened_coeffs) ** 2) /
np.mean(np.abs(flattened_coeffs) ** 2)
# Calculate zero crossing rate
features['wavelet_zero_crossing_rate'] =
np.mean(np.abs(np.diff(np.sign(flattened_coeffs))))
# Calculate peak count
features['wavelet_peak_count'] =
np.sum(np.diff(np.sign(np.diff(flattened_coeffs))) < 0)
 # Calculate inverse entropy
features['wavelet_inverse_entropy'] =
np.sum(np.abs(flattened_coeffs) *
np.log2(np.abs(flattened_coeffs) + 1e-12))
# Calculate coefficient range
features['wavelet_coefficient_range'] =
np.max(flattened_coeffs) - np.min(flattened_coeffs)
# Calculate spectral centroid and spread
spectral_centroids = [np.sum(np.arange(len(coeff)) *
np.abs(coeff) ** 2) / np.sum(np.abs(coeff) ** 2) if
np.sum(np.abs(coeff) ** 2) > 0
else 0 for coeff in coeffs]
spectral_spreads = [np.sqrt(np.sum(((np.arange(len(coeff))
- centroid) ** 2) * np.abs(coeff) ** 2) /
np.sum(np.abs(coeff) ** 2)) if np.sum(np.abs(coeff) ** 2)
> 0 else 0 for centroid, coeff in zip(spectral_centroids, coeffs)]
```

```
non_zero_centroids =
[centroid for centroid in spectral_centroids if centroid > 0]
non_zero_spreads =
[spread for spread in spectral_spreads if spread > 0]
mean_spectral_centroid = np.mean(non_zero_centroids) if
non_zero_centroids else 0
mean_spectral_spread = np.mean(non_zero_spreads) if non_zero_spreads
else 0
# Calculate weighted average of spectral centroids and spreads
features['wavelet_spectral_centroid'] =
mean_spectral_centroid
features['wavelet_spectral_spread'] = mean_spectral_spread
# Energy-related features
features['wavelet_energy_per_scale'] =
np.mean([np.sum(np.abs(coeff) ** 2) for coeff in coeffs])
features['wavelet_energy_ratio_per_scale'] =
np.mean([np.sum(np.abs(coeff) ** 2) /
total_energy for coeff in coeffs])
features['wavelet_cumulative_energy'] =
np.mean(np.cumsum([np.sum(np.abs(coeff) ** 2)
for coeff in coeffs]))
# Other coefficient-related features
features['wavelet_coefficient_mean_absolute_deviation'] =
np.mean([np.mean(np.abs(coeff - np.mean(coeff)))
for coeff in coeffs])
features['wavelet_coefficient_energy_ratio'] =
np.mean([np.sum(np.abs(coeff) ** 2) /
```

```
    total_energy for coeff in coeffs])
    all_features = {**time_features,
    **frequency_features, **features}
    return pd.DataFrame([all_features])
```

6.2.3 Random Forest Classifier

```
from sklearn.metrics import accuracy_score ,
confusion_matrix ,classification_report,
ConfusionMatrixDisplay
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier ,AdaBoostClassifier
from sklearn.svm import SVC ,LinearSVC
from sklearn.metrics import accuracy_score,
classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
import joblib
shuffled_df = duplicated_dataset.sample(frac = 1).
reset_index(drop=True)
x = shuffled_df.drop(["target"] ,axis = 1)
y = shuffled_df["target"]
x_train ,x_test ,y_train ,y_test =
train_test_split(x,y,test_size=0.05 ,random_state=42)
```

```
## scale the data
scaler = StandardScaler()  # or MinMaxScaler()
# Fit the scaler on the training data
and transform both training and testing data
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
joblib.dump(scaler, "scaler.joblib")
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(x_train, y_train)
# Evaluate Random Forest
rf_predictions = rf_model.predict(x_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
rf_classification_report =
classification_report(y_test, rf_predictions)
print("Random Forest Accuracy:", rf_accuracy)
print("Random Forest Classification Report:\n",
rf_classification_report)
# Save Random Forest model
joblib.dump(rf_model, "random_forest_model.joblib")
ConfusionMatrixDisplay(confusion_matrix(y_test,
rf_predictions)).plot()
```

6.2.4 Deep Neural Network Classifier

```
import tensorflow as tf
```

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(units = 128 ,
activation = "relu" ,input_shape=(237,)))
model.add(Dense(units = 128 ,activation = "relu"))
model.add(Dense(units = 128 ,activation = "relu"))
model.add(Dense(units = 64 ,activation = "relu"))
model.add(Dense(units = 32 ,activation = "relu"))
model.add(Dense(3 ,activation = "softmax"))
model.compile(optimizer='adam',
loss="categorical_crossentropy",
metrics = ["accuracy"])
model.summary()
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DeepneuralnetworkClassifier
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(x_train)
X_test_scaled = scaler.transform(x_test)
from keras.utils import to_categorical
y_train_encoded = to_categorical(y_train)
y_test_encoded = to_categorical(y_test)
model.fit(x_train ,y_train_encoded ,
batch_size = 28 ,epochs = 100)
joblib.dump(model, "deep_model.joblib")
y_predict_encoded = model.predict(x_test)
y_pred = np.argmax(y_predict_encoded,axis = 1)
```

```
y_pred
from sklearn.metrics import confusion_matrix,
accuracy_score,classification_report
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy)
print("\nClassification Report",
classification_report(y_test, y_pred))
```

6.2.5 Addition of noise PUF Vulnerability Analysis

```
import pandas as pd
file_path = r'C:\Users\Niraj\OneDrive - Wright State
University\Desktop\defence\CRP.xls'
crp_dataset = pd.read_excel(file_path)
# Display the first few rows of the dataset to
understand its structure
crp_dataset.head()
import numpy as np
# Rename columns for clarity:
Challenge bits (1-64) and Response
column_names = [f'bit_{i+1}' for i in range(64)]
+ ['response']
crp_dataset.columns = column_names
# Add Gaussian noise to the responses
noise_stddev = 0.009468699256063398
```

```
# Standard deviation of the Gaussian noise
crp_dataset['noisy_response'] = crp_dataset['response' +
np.random.normal(0, noise_stddev, len(crp_dataset))
# Display the first few rows of t
he updated datase to verify changes
crp_dataset.head()
# Specify the file path where you want
to save the updated dataset
save_file_path = r'C:\Users\Niraj\OneDrive - Wright State
University\Desktop\defence\CRP_Updated_
with_Noise_and_Renamed_Columns.xlsx'
# Save the dataset to an Excel file
crp_dataset.to_excel(save_file_path, index=False)
# If you see a FutureWarning regarding
the xlwt package, you might need to install openpyxl
and save as an .xlsx file
import pandas as pd
# Ensure the file path matches the file's actual format;
assuming it's '.xlsx' based on previous context
file_path = r'C:\Users\Niraj\OneDrive - Wright State
University\Desktop\defence\CRP_Updated_with
_Noise_and_Renamed_Columns.xlsx'
crp_dataset = pd.read_excel(file_path)
df = pd.read_excel(file_path)
# Display the first few rows of the dataset to
understand its structure
crp_dataset.head()
```

6.2.6 Random Forest Classifier For PUF Vulnerability Analysis

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
# Convert all column names to strings to
avoid the FutureWarning
df.columns = df.columns.astype(str)
# Creating interaction features (logical XOR
between pairs of the original features)
for i in range(0, 64, 2):
 df[f'xor_{i}_{i+1}'] = df.iloc[:, i] ^ df.iloc[:, i + 1]
# Selecting features and response variable
X = df.drop(columns=[df.columns[-1]])
# Exclude the last column which is the response variable
y = df.iloc[:, -1]  # The response variable
# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(X_scaled, y, test_size=0.16, random_state=0)
# Define the parameter grid for RandomForestClassifier
```

```
param_grid = {
   'n_estimators': [100, 200, 300],
   'max_depth': [None, 10, 20, 30],
   'min_samples_split': [2, 5, 10],
   'min_samples_leaf': [1, 2, 4],
   'bootstrap': [True, False]
}
# Create the GridSearchCV object for
RandomForestClassifier with 10-fold cross-validation
grid_search = GridSearchCV(estimator=
RandomForestClassifier(random_state=0),
                           param_grid=param_grid,
                           cv=10,  # Set to 10-fold cross-validation
                           n_jobs=-1,
                           verbose=2)
# Fit GridSearchCV
grid_search.fit(X_train, y_train)
# Get the best parameters and the best model
best_params = grid_search.best_params_
best_rf_model = grid_search.best_estimator_
print("Best Parameters for RandomForestClassifier:",
best_params)
# Predict with the best model
y_pred_best_rf = best_rf_model.predict(X_test)
# Calculate accuracy using the test data
accuracy_best_rf = accuracy_score(y_test, y_pred_best_rf)
print("Random Forest test accuracy with best parameters:", accuracy_best_rf)
```

REFERENCES

[1] A. Cirne, P. R. Sousa, J. S. Resende, and L. Antunes, "Hardware security for internet of things identity assurance," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2024.

[2] M. Núñez, N. T. Nguyen, D. Camacho, and B. Trawinski, "Computational collective intelligence," 2015.

[3] S. Rastayesh, "Risk assessment–with application for bridges and wind turbines," 2020.

[4] K. I. Gubbi, I. Kaur, A. Hashem, S. M. PD, H. Homayoun, A. Sasan, and S. Salehi, "Securing ai hardware: Challenges in detecting and mitigating hardware trojans in ml accelerators," in *2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2023, pp. 821–825.

[5] R. A. Meyers *et al.*, *Encyclopedia of complexity and systems science*. Springer New York, 2009, vol. 9.

[6] M. Haghbin, A. Sharafati, and D. Motta, "Prediction of channel sinuosity in perennial rivers using bayesian mutual information theory and support vector regression coupled with meta-heuristic algorithms," *Earth Science Informatics*, vol. 14, pp. 2279–2292, 2021.

[7] G. Chen, "Notice of retraction: The impact of information dimensionality on service system of e-commerce," in *2009 International Conference on Management and Service Science*. IEEE, 2009, pp. 1–4.

[8] ——, "Notice of retraction: The impact of information dimensionality on service system of e-commerce," in *2009 International Conference on Management and Service Science*, 2009, pp. 1–4.

[9] S. Torabi, A. Boukhtouta, C. Assi, and M. Debbabi, "Detecting internet abuse by analyzing passive dns traffic: A survey of implemented systems," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3389–3415, 2018.

[10] B. Holicza and A. Kiss, "Predicting and comparing students' online and offline academic performance using machine learning algorithms," *Behavioral Sciences*, vol. 13, no. 4, p. 289, 2023.

[11] R. Byali, "Using machine learning classifiers and a virtual voice assistant for common tasks, an employee performance evaluation model is used," *Journal homepage: www. ijrpr. com ISSN*, vol. 2582, p. 7421.

[12] A. Mahmoodzadeh, H. R. Nejati, and M. Mohammadi, "Optimized machine learning modelling for predicting the construction cost and duration of tunnelling projects," *Automation in Construction*, vol. 139, p. 104305, 2022.

[13] E. Gashaw, *Sesame Price Prediction Using Artificial Neural Network*. GRIN Verlag, 2020.

[14] K. I. Gubbi, B. Saber Latibari, A. Srikanth, T. Sheaves, S. A. Beheshti-Shirazi, S. M. PD, S. Rafatirad, A. Sasan, H. Homayoun, and S. Salehi, "Hardware trojan detection using machine learning: A tutorial," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 3, pp. 1–26, 2023.

[15] F. Farahmandi, M. S. Rahman, S. R. Rajendran, and M. Tehranipoor, *CAD for hardware security*. Springer, 2023.

[16] S. Akter, K. Khalil, and M. Bayoumi, "A survey on hardware security: Current trends and challenges," *IEEE Access*, 2023.

[17] Y. Obeng, C. Nolan, and D. Brown, "Hardware security through chain assurance," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1535–1537.

[18] S. Sasmal, "Securing data infrastructures with ai-powered solutions."

[19] T. Phan and R. Green, "Using wing flap sounds to distinguish individual birds," in *2023 IEEE International Conference on Electro Information Technology (eIT)*, 2023, pp. 083–094.

[20] E. Hossain, *Machine Learning Crash Course for Engineers*. Springer Nature, 2023.

[21] I. Sanchez-Gendriz, K. S. Azevedo, L. C. de Souza, M. G. Dalmolin, and M. A. Fernandes, "Gene sequence to 2d vector transformation for virus classification," *medRxiv*, pp. 2024–03, 2024.

[22] N. Thai-Nghe, Z. Gantner, and L. Schmidt-Thieme, "Cost-sensitive learning methods for imbalanced data," in *The 2010 International joint conference on neural networks (IJCNN)*. IEEE, 2010, pp. 1–8.

[23] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. Anwar, "Ton_iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems," *Ieee Access*, vol. 8, pp. 165 130–165 150, 2020.

[24] H. Qi, W. Liu, and L. Liu, "An efficient deep learning hashing neural network for mobile visual search," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2017, pp. 701–704.

[25] A. Černỳ, "Toxic content recognition in conversational systems," 2023.

[26] Y. Manzali and M. Elfar, "Random forest pruning techniques: a recent review," in *Operations research forum*, vol. 4, no. 2. Springer, 2023, p. 43.

[27] J. Bailey, L. Khan, and T. RW, *Advances in knowledge discovery and data mining*. Springer, 2016.

[28] M. Abu-Zanona, "Efficient iot security: Weighted voting for bashlite and mirai attack detection."

[29] Z. A. Ali, Z. H. Abduljabbar, H. A. Taher, A. B. Sallow, and S. M. Almufti, "Exploring the power of extreme gradient boosting algorithm in machine learning: A review," *Academic Journal of Nawroz University*, vol. 12, no. 2, pp. 320–334, 2023.

[30] T. Miller, K. Lewita, A. Krzemińska, P. Kozlovska, M. Jawor, D. Cembrowska-Lech, and A. Kisiel, "Boosting modern society: advancements and applications of the adaboost algorithm in diverse domains," *Scientific Collection InterConf*, no. 152, pp. 549–555, 2023.

[31] B. Stanoev, G. Mitrov, A. Kulakov, G. Mirceva, P. Lameski, and E. Zdravevski, "Automating feature extraction from entity-relation models: Experimental evaluation of machine learning methods for relational learning," *Big Data and Cognitive Computing*, vol. 8, no. 4, p. 39, 2024.

[32] H. MaBouDi, A. B. Barron, S. Li, M. Honkanen, O. J. Loukola, F. Peng, W. Li, J. A. Marshall, A. Cope, E. Vasilaki *et al.*, "Non-numerical strategies used by bees to solve numerical cognition tasks," *Proceedings of the Royal Society B*, vol. 288, no. 1945, p. 20202711, 2021.

[33] D. Griner, *The Development and Optimization of a Deep-Learning Strategy for COVID-19 Classification in Chest X-Ray Radiography*. The University of Wisconsin-Madison, 2023.

[34] J. Naskath, G. Sivakamasundari, and A. A. S. Begum, "A study on different deep learning algorithms used in deep neural nets: Mlp som and dbn," *Wireless personal communications*, vol. 128, no. 4, pp. 2913–2936, 2023.

[35] H. Cheng, D. Lian, S. Gao, and Y. Geng, "Utilizing information bottleneck to evaluate the capability of deep neural networks for image classification," *Entropy*, vol. 21, no. 5, p. 456, 2019.

[36] F. de-la Calle-Silos, A. Gallardo-Antolín, and C. Peláez-Moreno, "Deep maxout networks applied to noise-robust speech recognition," in *Advances in Speech and Language Technologies for Iberian Languages: Second International Conference, IberSPEECH 2014, Las Palmas de Gran Canaria, Spain, November 19-21, 2014. Proceedings*. Springer, 2014, pp. 109–118.

[37] P. Rauber, "Visual analytics applied to image analysis," *Proceedings of the Visual Analytics Science and Technology*, vol. 23, no. 01, p. 2017, 2016.

[38] S. Lohmüller, "Cognitive self-organizing network management for automated configuration of self-optimization son functions," Ph.D. dissertation, Universität Augsburg, 2019.

[39] N. T. Nguyen, R. Chbeir, E. Exposito, P. Aniorté, and B. Trawiński, *Computational Collective Intelligence: 11th International Conference, ICCCI 2019, Hendaye, France, September 4–6, 2019, Proceedings, Part II*. Springer Nature, 2019, vol. 11684.

[40] J. Emery, "Construction and statistical analysis of an industry wide ground control database of mechanical roof extensometer data from underground coal mine gateroads," Ph.D. dissertation, UNSW Sydney, 2024.

[41] R. Nouri, "The need to go deeper: the employment of a convolutional neural network to analyze turbulent flows frequency content," Ph.D. dissertation, Tennessee Technological University, 2023.

[42] P. Liu, B. Qian, Q. Sun, and L. Zhao, "Prompt-wnqa: A prompt-based complex question answering for wireless network over knowledge graph," *Computer Networks*, vol. 236, p. 110014, 2023.

[43] A. Alzahrani and M. Z. Asghar, "Intelligent risk prediction system in iot-based supply chain management in logistics sector," *Electronics*, vol. 12, no. 13, p. 2760, 2023.

[44] J.-S. Park, J.-J. Lee, Y.-J. Choi, T.-W. Moon, S. Kim, S. Cho, H. Kang, D. H. Kim, J. Park, and S.-W. Choi, "Physical unclonable functions employing circularly polarized light emission from nematic liquid crystal ordering directed by helical nanofilaments," *ACS Applied Materials & Interfaces*, 2024.

[45] S. Hemavathy and V. K. Bhaaskaran, "Arbiter puf-a review of design, composition, and security aspects," *IEEE Access*, 2023.

[46] M. Ferens, E. Dushku, and S. Kosta, "Ml for attack and defense of pufs: Current status and future directions," in *International Symposium on Distributed Computing and Artificial Intelligence*. Springer, 2023, pp. 389–398.

[47] J. Vosatka, "Introduction to hardware trojans," *The Hardware Trojan War: Attacks, Myths, and Defenses*, pp. 15–51, 2018.

[48] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," *Computer*, vol. 43, no. 10, pp. 39–46, 2010.

[49] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*. Springer, 1999, pp. 388–397.

[50] Y. Zhao, S. Pan, H. Ma, Y. Gao, X. Song, J. He, and Y. Jin, "Side channel security oriented evaluation and protection on hardware implementations of kyber," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.

[51] N. Q. M. Noor, N. N. A. Sjarif, N. H. F. M. Azmi, S. M. Daud, and K. Kamardin, "Hardware trojan identification using machine learning-based classification," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, no. 3-4, pp. 23–27, 2017.

[52] H. Wang, G. Zhao, S. Lu, L. Li, W. Zhang, and J. Liu, "Investigation on hydrocarbon generation and expulsion potential by deep learning and comprehensive evaluation method: A case study of hangjinqi area, ordos basin," *Marine and Petroleum Geology*, vol. 144, p. 105841, 2022.

[53] W. Xu, L. Pang, Y. Tang, and M. Chen, "Security evaluation of feed-forward interpose puf against modelling attacks," in *2024 IEEE 4th International Conference on Power, Electronics and Computer Applications (ICPECA)*. IEEE, 2024, pp. 871–877.

[54] X. Wang, J. Li, X. Kuang, Y.-a. Tan, and J. Li, "The security of machine learning in an adversarial setting: A survey," *Journal of Parallel and Distributed Computing*, vol. 130, pp. 12–23, 2019.

[55] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," *arXiv preprint arXiv:1810.00069*, 2018.

[56] S. Faezi, R. Yasaei, A. Barua, and M. A. Al Faruque, "Brain-inspired golden chip free hardware trojan detection," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2697–2708, 2021.

[57] F. Farahmandi, Y. Huang, and P. Mishra, *System-on-chip security*. Springer, 2020.

[58] C. B. C. d. Cunha, T. A. Lima, D. L. d. M. Ferraz, I. T. C. Silva, M. K. D. Santiago, G. R. Sena, V. S. Monteiro, and L. B. Andrade, "Predicting the need for blood transfusions in cardiac surgery: A comparison between machine learning algorithms and established risk scores in the brazilian population," *Brazilian Journal of Cardiovascular Surgery*, vol. 39, p. e20230212, 2024.

[59] J. Barrera-García, F. Cisternas-Caneo, B. Crawford, M. Gómez Sánchez, and R. Soto, "Feature selection problem and metaheuristics: A systematic literature review about its formulation, evaluation and applications," *Biomimetics*, vol. 9, no. 1, p. 9, 2023.

[60] A. Bailly, "Time series classification algorithms with applications in remote sensing," Ph.D. dissertation, Université Rennes 2, 2018.

[61] A. C. Ferreira, L. R. Silva, F. Renna, H. B. Brandl, J. P. Renoult, D. R. Farine, R. Covas, and C. Doutrelant, "Deep learning-based methods for individual recognition in small birds," *Methods in Ecology and Evolution*, vol. 11, no. 9, pp. 1072–1085, 2020.

[62] M. M. Abbassy, "Using machine learning technique for analytical customer loyalty," *International Journal of Computer Science & Network Security*, vol. 23, no. 8, pp. 190–198, 2023.

[63] K. Arai, *Advances in Information and Communication: Proceedings of the 2022 Future of Information and Communication Conference (FICC), Volume 2*. Springer Nature, 2022, vol. 439.

[64] E. L. Viganò, D. Ballabio, and A. Roncaglioni, "Artificial intelligence and machine learning methods to evaluate cardiotoxicity following the adverse outcome pathway frameworks," *Toxics*, vol. 12, no. 1, p. 87, 2024.

[65] S. Chen, T. Wang, Z. Huang, and X. Hou, "Detection method of hardware trojan based on attention mechanism and residual-dense-block under the markov transition field," *Journal of Electronic Testing*, vol. 39, no. 5, pp. 621–629, 2023.