UNIVERSIDADE D
COIMBRA

António Pedro Correia

AI-Based Intrusion Detection Mechanisms for Cloud-native Services

Dissertation in the context of the Masters in Informatics Security, advised by Professor Naghmeh Ramezani Ivaki and Dr. Paulo Miguel Guimarães da Silva and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2024

DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

António Pedro Correia

AI-Based Intrusion Detection Mechanisms for Cloud-native Services

Dissertation in the context of the Masters in Informatics Security, advised by Professor Naghmeh Ramezani Ivaki and Dr. Paulo Miguel Guimarães da Silva and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2024

DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

António Pedro Correia

Mecanismos de deteção de intrusão baseados em IA para serviços Cloud

Dissertação no âmbito do Mestrado em Segurança Informática, orientada pela Professora Naghmeh Ramezani Ivaki e pelo Doutor Paulo Miguel Guimarães da Silva e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho 2024

This work was carried out in the scope of the Agenda "NEXUS - Pacto de Inovação – Transição Verde e Digital para Transportes, Logística e Mobilidade", financed by the Portuguese Recovery and Resilience Plan (PRR), with no. C645112083-00000059 (investment project no. 53)

Acknowledgements

I would like to express my gratitude to Professor Naghmeh Ramezani Ivaki and Dr. Paulo Miguel Guimarães da Silva for their guidance, support, and feedback throughout the entire process of conducting this research and development. Their expertise, encouragement, and patience have been fundamental.

I am also deeply thankful to the entire team at Instituto Pedro Nunes for providing a conducive and stimulating research environment. The resources and opportunities provided to me significantly contributed to the successful completion of this work.

I would also like to give a special thanks to the Department of Informatics Engineering at the Faculty of Sciences and Technology of the University of Coimbra. The academic atmosphere and collaborative environment within the department have been essential during my whole academic progress.

I extend my appreciation to the members of the jury, Carlos Ribeiro and Tiago Cruz, for their time, expertise, and feedback during the evaluation process.

To my friends and family, thank you for your support and understanding throughout my whole academic journey. Your encouragement has been a source of strength and motivation.

Finally, I would like to express my gratitude to anyone who, unbeknownst to me, played a role in the development of this thesis, and whose contributions may have occurred behind the scenes.

This research would not have been possible without the effort or presence of all those mentioned above, both known and unknown. I am thankful for the diverse contributions that have made this work possible.

Abstract

This thesis focuses on developing an effective machine-learning solution tailored for cloud applications, specifically targeting the detection of anomalies through environmental feature analysis. Traditional security techniques often fail due to the rapid and dynamic nature of cloud-native systems, highlighting the necessity for cloud-specific strategies. To address this, models including Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Gaussian Naive Bayes (GNB), and Random Forest were rigorously evaluated. A significant emphasis was placed on feature engineering to enhance model performance. The findings show the critical role of feature selection in constructing robust machine-learning models, demonstrating how varying objectives may benefit from distinct feature selection methodologies. This document comprehensively outlines the objectives of the internship, the methodology employed to achieve these goals, and provides a detailed analysis and discussion of the testing results. It highlights the specific advantages of each machine-learning model tested and offers insights into their relative effectiveness in a cloud context. Additionally, the document discusses the practical implications of these findings for the development and deployment of security measures in cloud environments. By presenting a thorough examination of feature engineering techniques and their impact on model performance, this work contributes to the field of cloud security. The research illustrates the potential of machine learning in enhancing cloud security, showcasing the benefits of tailored feature selection strategies in optimizing model accuracy, reliability, and performance.

Keywords

Cloud Security, Intrusion Detection Systems, Artificial Intelligence, Machine Learning, Feature Engineering

Resumo

Este documento foca-se no desenvolvimento de uma solução eficaz de machine-learning, desenvolvida para aplicações em cloud, direcionada para a deteção de anomalias através da análise de features extraidas destes ambientes. Os métodos de segurança tradicionais frequentemente falham devido à natureza rápida e dinâmica dos sistemas nativos de Cloud, mostrando a necessidade de estratégias desenvolvidas especificamente para estes ambientes. Para abordar este problema, foram rigorosamente avaliados os seguintes modelos: Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Gaussian Naive Bayes (GNB), e Random Forest. Foi dada uma ênfase significativa ao processo de feature engineering para melhorar o desempenho dos modelos. Os resultados mostram a importância da seleção de características no desenvolvimento de modelos robustos, demonstrando como diferentes objetivos podem beneficiar de metodologias distintas de feature selection. Este documento aborda os objetivos deste estágio, a metodologia empregue para alcançar estes objetivos e fornece uma análise detalhada e discussão dos resultados dos testes. Realça também as vantagens específicas de cada modelo testado e analisa a sua efetividade relativa num contexto de cloud. Adicionalmente, o artigo discute as implicações práticas destes resultados para o desenvolvimento e implementação de medidas de segurança em ambientes de cloud. Ao apresentar um exame aprofundado das técnicas de engenharia de características e o seu impacto no desempenho dos modelos, este trabalho contribui para o campo da segurança em Cloud. A pesquisa ilustra o potencial de machine-learning em melhorar a segurança na Cloud, evidenciando os benefícios de estratégias de feature selection na otimização da precisão, fiabilidade e desempenho dos modelos.

Palavras-Chave

Segurança em Cloud, Sistemas de Detecção de Intrusão, Inteligência Artificial, Machine Learning, Feature Engineering

Contents

Acronyms

AI Artificial Inteligence.

CASB Cloud Access Security Broker.

CC Cloud Computing.

DDoS Distributed Denial of Service.

DoS Denial of Service.

GDPR General Data Protection Regulation.

GNB Gaussian Naïve Bayes.

HIDS Host-Based Intrusion Detection System.

HIPAA Health Insurance Portability and Accountability Act.

IaaS Infrastructure as as Service.

IAM Identity and Access Management.

IDS Intrusion Detection System.

IT Information Technology.

KNN K Nearest Neighbour.

MFA Multi-factor Authentication.

ML Machine Learning.

NIDS Network-Based Intrusion Detection System.

PaaS Platform as a Service.

PCI DSS Payment Card Industry Data Security Standard.

PII Personally Identifiable Information.

SaaS Software as a Service.

SIEM Security Information and Event Management.

SVM Support Vector Machine.

VPC Virtual Private Cloud.

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides the context and motivation for this research. It also identifies the problem to be addressed and the objectives to be followed to develop a solution for that problem. Finally, the structure of the document is presented.

1.1 Context and Motivation

In an era marked by the growth of digital evolution, the current increase in cyber threats is impossible to overlook. Systems are moving towards cloud-based solutions, offering resource-sharing and access to powerful computing resources even for those with rudimentary setups. Yet, this transition does not come without downsides, exposing systems to higher risks and fueling the rise of cybercrime, whose costs are skyrocketing, growing 15 percent each year, and expected to reach a staggering 10.5 trillion USD by 2025 [Morgan, 2020].

To counter this growth, Artificial Inteligence (AI), with a particular emphasis on the capabilities of Machine Learning (ML), is an essential tool that should be taken in as another weapon against cybercrime. Computers, with the cutting edge of AI capabilities, outshine their human counterparts when it comes to pattern recognition, which is an indispensable skill in cyber attack detection. Moreover, these AI-driven systems operate without being subjected to fatigue, erasing another cause of concern that is highly pronounced in human vigilance, turning them into the best guardians for Information Technology (IT) systems. This high-level defense mechanism is possible due to ML, a subset of AI that not only learns from vast datasets but also adeptly identifies anomalies and dynamically adapts to the environment.

The motivation to explore and enhance security mechanisms within cloud-native services comes from the necessity to preserve confidentiality, integrity, and availability of digital assets in an environment characterized by its dynamicity and complexity. Traditional security paradigms, often designed for static architectures, might become outpaced by the fast dynamism of a cloud-native system, resulting in some cloud-specific attacks that might cut through those defenses, which poses a significant risk. In the context of the cloud, security extends be-

yond a mere concern. It becomes an essential element in organizational risk management and strategic planning. While preventative measures remain indispensable, there are always more innovative attackers, making intrusion detection an essential defense in the world of cybersecurity.

Integrating AI and intrusion detection within cloud-native environments marks a significant advancement in cybersecurity capabilities. As mentioned before, the capability of a ML algorithm to learn from vast datasets and adapt to their surroundings makes it a formidable asset in security in general. As organizations progress through their digital transformations into the cloud, the integration of this technology becomes a great response to the escalation of problems that come with it.

The primary purpose of this work is to study, develop, and propose a set of machine-learning solutions that can accurately identify intrusions with data extracted from the cloud system in which it is deployed, such as container CPU usage, packet volume, etc. This solution will be developed with cloud-native services in mind, so it will be tailored for this purpose to make it more efficient than traditional solutions when applied to these scenarios. By leveraging the dynamic and contextual information provided by container and service deployment metrics, such as the metrics that were previously mentioned, it is theoretically possible to develop intelligent intrusion detection systems capable of detecting attacks without the need for an extensive set of features, as shown by the results of this thesis.

This work has been supported by Project "Agenda Mobilizadora Sines Nexus". ref. No. 7113), supported by the Recovery and Resilience Plan (PRR) and by the European Funds Next Generation EU, following Notice No. 02/C05-i01/2022, Component 5 - Capitalization and Business Innovation - Mobilizing Agendas for Business Innovation.

1.2 Problem Statement

The widespread adoption of cloud-native architectures and containerization has revolutionized application deployment. However, this paradigm brings a new set of security challenges, emphasizing the need for effective Intrusion Detection Systems (IDS). Traditional approaches rely on signature-based or rule-based detection methods, which often struggle to keep up with rapidly evolving attack vectors. Cloud-native environments often involve complex architectures with numerous interconnected components and shared responsibilities. To address this limitation, leveraging Artificial Intelligence (AI) techniques can enhance intrusion detection by automatically learning patterns and detecting anomalies in real time.

Intrusion detection is crucial in cloud services due to cloud environments' dynamic and scalable nature, which often handle vast amounts of sensitive data and critical applications. Cloud services are more susceptible to a broader range of threats and attacks than traditional IT infrastructures due to their distributed

architecture and the shared responsibility model between service providers and clients. This thesis aims to explore AI-based intrusion detection mechanisms for effective and efficient detection of anomalies in cloud-native environments.

1.3 Objectives

The objectives aimed to achieve with this work are the following:

(i) Analyze the state of the art on ML-Based Intrusion Detection Systems: To firmly understand the current machine learning-based solutions for intrusion detection, analyzing the state of the art on Intrusion Detection Systems is the first step. This is one of the most important parts, as it will define the work baseline for the future steps.

(ii) Machine Learning Models research and experimentation: After the analysis of the state of the art, experimenting with diverse models, such as SVM, KNN, GNB, and Random Forest, to recreate the results of other papers is essential, as usually no code is provided for the presented solutions.

(iii) Feature Engineering: After implementing the models, their performance needs to be improved. This can be achieved by altering the features on the dataset, as some of them can do more harm than good.

(iv) Implementation of the selected ML Models on data from other datasets: Even if the improvement of the models is successful in the previous objective, the models might not be usable with the metrics that are extracted from the system, as the features are much more limited than the ones used in typical training datasets. Using other datasets might show the performance of the models when dealing with other types of features.

(v) Implement and test a functional prototype: The next step is to implement a functional prototype and test it using any models that proved useful in the previous objective and that can accurately identify and report intrusions with the extracted data.

(vi) Evaluate the solution and document the results: After the implementation, we have the evaluation of the solution and the documentation of the results obtained. It is important to conclude whether the mechanisms were successful and whether the feature selection process affects them.

1.4 Document Structure

This thesis consists of eight chapters. The **first chapter** is the introduction. The first section of the chapter presents the context and motivation behind this thesis. The second section is about the problem being addressed, followed by the third section, which presents the objectives that need to be achieved. The last section, this one, is to address how the document is structured.

The **second chapter** is for the Background, information that is related to the topic, which can be helpful for the reader of the thesis. The sections of this chapter are pretty straightforward: the first section is related to cloud security in general, the second one addresses intrusion detection systems, the third one is for machine learning applied to intrusion detection systems, and the fourth and final one for feature extraction.

The **third chapter** addresses works similar to what was done in this thesis, which served as inspiration, guidance, and learning material. It focuses on the procedures used to obtain such documents and goes over which areas of cloud security are addressed. It then documents the models, datasets, and evaluation metrics that were used in those papers.

The **fourth chapter** focuses on the approach used. The first section is for the expected outcome, and the second section outlines the proposed approach to reach that outcome. The third section features the methodology that was used concerning the model and dataset selection, the evaluation and selection of the features, the two methods used to train and evaluate the models, the metrics used to quantify those evaluations, and the tools and frameworks that were used.

The **fifth chapter** presents the results obtained from the first method. The first section presents the results of the selection of the models and the dataset, while the second one specifies how those were implemented and validated. The third section provides the results of the process of feature engineering. The fourth section presents some conclusions taken from those results and highlights the limitations of the work done.

The **sixth chapter** is about the second and final method. The first section presents the newly introduced datasets, the second shows the extensive results of this method, separating them by dataset and model, and the third section discusses those results and addresses their limitations.

The **seventh and final chapter** addresses the end results of this thesis, therefore, concluding it.

Chapter 2

Background

This section aims to provide foundational knowledge about the topics that are addressed in this thesis. To better understand the foundations of the area, some research about basic topics was done, as part of the literature review portion of the work. An in-depth analysis of cloud security, intrusion detection systems, and machine learning is presented. A section about feature extraction is also provided, in which the work that was done to extract the features from the containerized system will be presented.

2.1 Cloud Security

With the constant evolution of digital transformation, Cloud Computing (CC) has become an extremely important beacon in the way businesses operate and users interact with Information Technology (IT). As organizations migrate their data and applications to the cloud, the importance of ensuring their safety has never been more critical. CC is a paradigm that involves the availability of computing services, such as storage, processing power, and applications, over the internet [Mell and Grance, 2011]. Service models such as Infrastructure as as Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) form the foundation of CC. The shift to cloud computing has revolutionized the way organizations operate, providing agility, scalability, and cost-efficiency [Armbrust et al., 2010]. Unfortunately, this shift also introduces new security challenges, as sensitive data and critical applications are no longer housed within the usual on-premises infrastructure. Cloud security is imperative to protect against a wide range of threats, including unauthorized access, data breaches, and service disruptions. Additionally, compliance with regulatory requirements further underscores the significance of robust cloud security practices. Figure 1 shows the typical architecture of cloud computing.

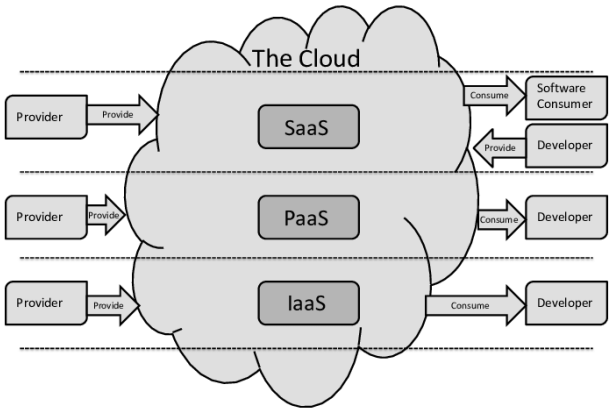


Figure 1: Cloud Computing Architecture [Muchahari and Sinha, 2013]

2.1.1 Challenges

One of the primary challenges in cloud security is the risk of data breaches and unauthorized access [Institute, 2022], since the data is being stored in remote servers, heightening the potential for malicious actors to gain access to it. This is due to the shared responsibility model, wherein both the cloud service provider and the customer have specific security responsibilities, adding complexity to mitigating these risks [inc, 2018]. Unauthorized access may lead to data exfiltration, identity theft, or the compromise of critical business information.

Ensuring compliance with industry regulations and legal requirements also poses another significant challenge in cloud security [PARLIAMENT, 2016]. Different sectors and regions have varying compliance standards and organizations must take on the complex task that is complying with the vast landscape of regulations, such as General Data Protection Regulation (GDPR), Health Insurance Portability and Accountability Act (HIPAA), and Payment Card Industry Data Security Standard (PCI DSS). Failing to meet these standards can result in severe legal consequences, financial penalties, and damage to the organization's reputation. Cloud Security is also intricately linked to the concept of data residency and sovereignty. Organizations must grapple with questions of where their data is physically stored and which jurisdiction's laws apply to it. This is particularly important in cases where sensitive data, such as Personally Identifiable Information (PII), is subject to stringent data protection regulations. Just like compliance, navigating these challenges requires a nuanced understanding of legal frameworks and careful consideration of data governance policies.

More conventional security challenges, such as attacks like Distributed Denial of Service (DDoS), malware injection, and many others, are also present in CC. Although these might be harder to perform due to the cloud's dynamic nature and encrypted traffic, they are still possible, and these characteristics that make it harder to attack are also what makes it a double-edged sword, also making the process of detection difficult. Fortunately, some tools can be integrated into the system that makes the detection process easier, which will later be mentioned.

2.1.2 Strategies

Encryption is a cornerstone of cloud security, offering a robust defense against unauthorized access and data breaches. By encrypting data both in transit and at rest, organizations can ensure that even if a breach occurs, the intercepted data remains unreadable without the corresponding decryption keys. Strong encryption algorithms and securely stored encryption keys both contribute to an effective encryption strategy.

Controlling access to resources and data is also fundamental to cloud security [Alliance, 2017]. Identity and Access Management (IAM) solutions enable organizations to manage user identities, control access permissions, and enforce authentication mechanisms. Implementing the principle of least privilege also helps ensure that users only have access to the resources necessary for their respective roles, minimizing the potential impact of compromised credentials. Multi-factor Authentication (MFA) further enhances this aspect, requiring users to provide multiple forms of verification before letting them access the data.

Network security also plays a pivotal role in securing cloud environments. Virtual Private Cloud (VPC), firewalls, and intrusion detection systems help create secure network architectures. Network segmentation ensures that even if an attacker gains access to one segment, they are still isolated from other parts of the network. This containment strategy limits the lateral movement of threats, reducing the overall risk of a security incident. Proactive Monitoring of cloud environments is crucial for detecting and responding to security incidents in real-time [Dempsey et al., 2011]. Security Information and Event Management (SIEM) solutions, combined with advanced threat detection mechanisms, provide organizations with the capability to identify suspicious activities and potential breaches. A well-defined incident response plan ensures that in the event of a security incident, the organization can respond swiftly, minimizing the impact and restoring normal operations. Regular audits, however, can reduce the need for such actions, as they evaluate the effectiveness of the cloud security measures, exposing many of the existing flaws present in the system. Conducting thorough security assessments, vulnerability scans, and penetration testing helps identify possible entry points for attackers. Compliance checks ensure that the organization remains in adherence to industry regulations and standards.

2.1.3 Technologies Shaping Cloud Security

Many technologies are gaining prominence in the cloud security realm, such as the Zero Trust model which is a paradigm shift in security strategy where, unlike traditional approaches, it is assumed that threats may exist both outside and inside the network, advocating for strict access controls, continuous verification and the previously mentioned principle of least privilege [of Standards and Technology, 2020]. The architecture of zero trust is shown in Figure 2. Implementing this architecture in a cloud system involves robust authentication mechanisms, micro-segmentation, and continuous monitoring to ensure the security of every interaction, regardless of the user's location or device.

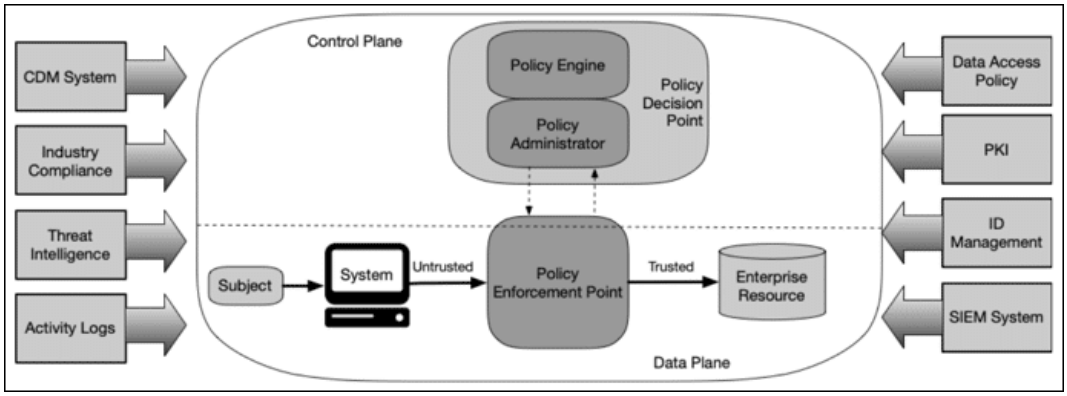


Figure 2: Zero Trust Architecture [Rose et al., 2020]

Another technology is Cloud Access Security Broker (CASB), acting as an intermediary between cloud service users and cloud applications, providing an additional layer of security and visibility [C. Riley, 2020]. CASBs help organizations enforce security policies, monitor user activity, and protect data in real-time. They play a crucial role in addressing the challenges associated with the shared responsibility model, offering organizations greater control over their security approach in the cloud. The workflow for this technology can be seen in Figure 3.

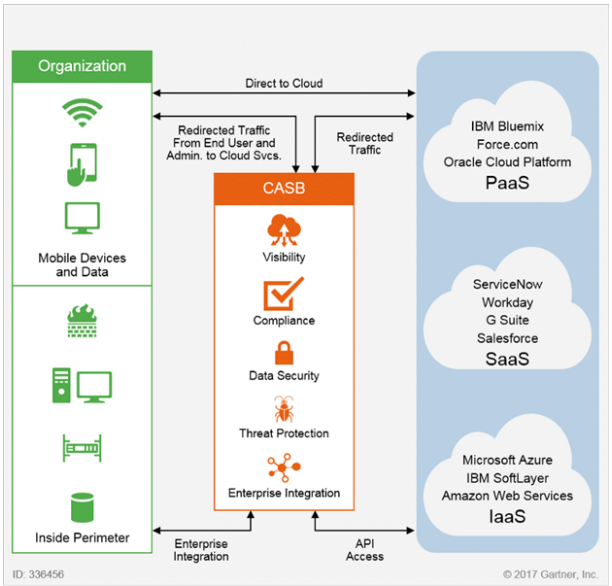


Figure 3: Cloud Access Security Broker Workflow [BasuMallick, 2022]

The adoption of containerization technologies, such as Docker and Kubernetes, has introduced new considerations for cloud security. They provide a lightweight and scalable way to package and deploy applications [Docker, 2020]. However, container security involves securing the entire container lifecycle, from image creation to runtime. Implementing container security best practices, such as image scanning, runtime protection, and access control, is essential to prevent any vulnerability and ensure the security of containerized applications in the cloud.

Serverless computing, where applications run in ephemeral containers without

the need for managing servers, offers scalability and cost benefits, but organizations must address security concerns related to code execution, data storage, and event triggers [Services, 2021]. Code analysis and runtime protection are common security measures for serverless environments that should not be skipped.

2.1.4 Emerging Trends

Artificial Inteligence (AI) and Machine Learning (ML) are becoming integral components of cloud security. These technologies enable organizations to analyze vast amounts of data, detect anomalies, and identify potential threats in real-time. AI and ML-powered security solutions, such as the one developed with the support of this thesis, enhance the ability to predict, prevent, and respond to security incidents, offering a proactive approach to cloud security.

2.2 Intrusion Detection Systems

Intrusion Detection Systems (IDS) play a crucial role in safeguarding computer networks and systems from unauthorized access, malicious activities, and cyber threats [Roesch, 1999]. These systems actively monitor and analyze network traffic, providing a critical layer of protection.

2.2.1 Core Functionalities

An IDS serves as a vigilant gatekeeper, using various methodologies to identify potential threats in the systems they are assigned to. They engage in thorough packet analysis, scrutinizing the contents of the network packets for suspicious patterns or anomalies. Log analysis can also be conducted on logs generated by network devices to discern unusual patterns or security events. Operating in real-time, an IDS continuously monitors network traffic, enabling prompt detection and response to potential threats with immediate alerts or automated actions based on predefined rules. These functionalities collectively create a comprehensive defense mechanism, allowing the IDS to identify both known and unknown threats, ensuring swift responses to deviations from normal behavior.

2.2.2 Types of Intrusion Detection Systems

IDSs come in two primary types: Network-Based Intrusion Detection Systems (NIDS) and Host-Based Intrusion Detection Systems (HIDS). NIDS are strategically positioned on the network and inspect all incoming and outgoing packets, identifying potential threats, while HIDS, on the other hand, are installed on individual hosts or servers, monitoring activities on the host level, analyzing system logs, file integrity, and user behavior to detect signs of compromise. NIDS provides a comprehensive view of network traffic, making it effective in identifying

threats that traverse multiple hosts, acting as a proactive guardian at the network perimeter, and ensuring the detention of potentially harmful activities, before they reach individual hosts. Conversely, HIDS focuses on specific activities that might evade network-level detection. Combining both types is often referred to as Hybrid IDS, offering organizations a multi-layered defense strategy.

IDSs are also split into two types of deployment, Signature-Based IDS (also known as Knowledge-Based IDS or Misuse IDS) and Anomaly-Based IDS, as seen in Figure 4. The first option relies on a database of already known attack signatures, effectively identifying documented threats. This makes it particularly efficient in recognizing well-established and widely documented threats, however, it may struggle with detecting newer and unknown threats that do not match any already existing signature. Anomaly-Based IDS, on the other hand, establishes a baseline of normal network behavior and triggers alerts when deviations occur. This approach fills the holes of the previous one, being efficient at detecting zero-day attacks, as it does not rely on pre-defined signatures. This type of detection is crucial for identifying threats that signature-based systems may miss, however, it may also generate false positives, triggering alerts for non-malicious activities that deviate from the established baseline. For this reason, it is always best to leave the final decision to a system administrator. A hybrid solution is also possible, combining both signature-based and anomaly-based detection methods, offering a comprehensive approach that leverages the strengths of both strategies to enhance detection accuracy. By combining the efficiency of signature-based detection for known threats and the adaptability of anomaly-based detection for unknown threats, organizations can achieve a more accurate IDS solution.

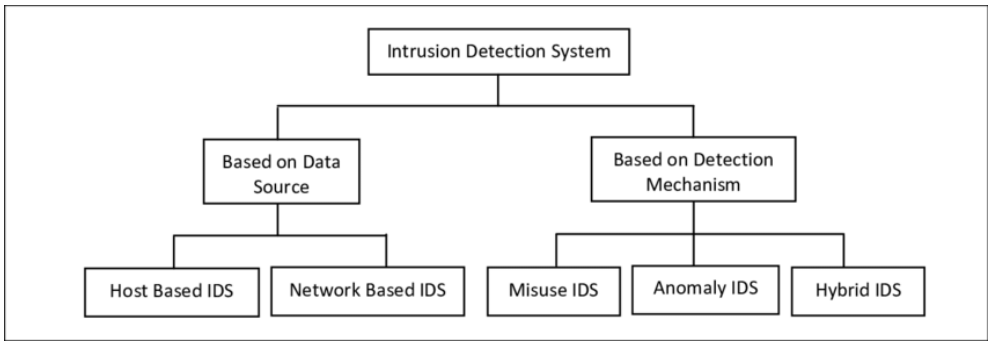


Figure 4: Types of IDS [Sulaiman et al., 2021]

2.2.3 Challenges in Intrusion Detection Systems

While IDS are pivotal for network security, they encounter challenges that must be mentioned. False positives, for example, where normal activities are mistakenly flagged as suspicious, can lead to unnecessary alerts and potential alert fatigue. It is important to fine-tune the IDS to minimize the number of false positives, without compromising its ability to detect actual threats.

Encrypted traffic also poses a challenge for IDS, as inspecting the contents of encrypted packets becomes difficult. Encryption ensures the privacy and integrity

of data in transit, but it also creates blind spots for traditional IDS. This is especially relevant in Cloud environments. In this context, an IDS also faces diverse challenges such as the dynamic nature of cloud resources, the scalability of services, and much more. Traditional methods may struggle to accompany the fast pace of the deployment and reconfiguration of cloud instances, so it is important to experiment with different strategies to check what best fits the scenario.

2.3 Machine Learning in Intrusion Detection Systems

As mentioned before, as the complexity of cyber attacks evolves, traditional IDS solutions are often insufficient. This has led to the integration of ML techniques in intrusion detection, offering enhanced capabilities to identify and respond to threats. In this section, the application of ML in IDSs will be explored, as well as its impact on improving the overall security of modern IT infrastructures.

2.3.1 Role and Benefits of Machine Learning

ML techniques play a pivotal role in enhancing the efficiency and effectiveness of IDSs. Traditionally, IDSs rely on rule-based methods, which struggle to adapt to the dynamic evolution of cyber threats. ML, however, enables systems to learn from data patterns and autonomously identify anomalies. Algorithms, such as neural networks and decision trees, analyze vast datasets to discern normal network behavior and detect deviations that indicate potential intrusions [Talukder et al., 2024]. ML also contributes to the scalability of an IDS. As the volume and complexity of data generated in a network continue to grow, ML algorithms excel at handling vast amounts of information, allowing for timely and accurate detection of both known and unknown threats [Liu and Lang, 2019].

The adoption of ML in IDS brings a lot of benefits with it. One notable advantage is the improved accuracy of threat detection. ML algorithms excel at recognizing patterns and anomalies that might escape the traditional solutions. This accuracy translates into fewer false positives and negatives, providing security teams and admins with more reliable information [Bakhsh et al., 2023]. The speed of threat detection is also enhanced, therefore also increasing the speed of response. The natural speed of these systems allows them to quickly identify, and sometimes even mitigate, security incidents in real-time, minimizing the potential damage caused by attacks [Thakkar and Lohiya, 2023]. ML also facilitates the adaptability of the IDS to specific environments. Unlike rigid rule-based systems, ML models can adapt to the unique characteristics of different networks, ensuring that the IDS is tailored to the specific needs and vulnerabilities of an organization.

2.3.2 Challenges and Considerations

While ML-based IDSs offer significant advantages, they also come with challenges. A big challenge lies in the interpretability of the models within the IDS.

Understanding how a model reaches a specific decision is crucial for trust and accountability. The opacity of certain ML algorithms can make it challenging to interpret and validate the reasoning behind a detected threat, which is why eXplainable AI, a research field that aims to make AI systems more understandable to humans, is gaining popularity. There are also privacy concerns that arise when implementing ML in IDS, particularly in environments with sensitive data. The need to collect and analyze network traffic data to train ML models may conflict with privacy regulations or organizational policies.

Moreover, the quality of the data is also extremely important. Ensuring that the data used to train ML is representative and unbiased is essential for the system's accuracy and reliability. Oversampling, for example, can easily introduce biases. When certain classes are overrepresented, the model may become skewed towards them, potentially leading to wrong predictions in threat detection. If the training data lacks diversity, the model might also struggle to generalize effectively to new and unseen threats. Additionally, the accuracy of previously obtained data might pose challenges in handling the threats. If the data fed to the model was obtained from previous versions of the system that had a slightly different behavior, the predictions might become incorrect, as the normal behavior of the system would differ from the more recent one.

2.4 Feature Extraction Tools

Metric extraction tools are indispensable to an IDS, as these are what allow it to receive the information needed for the predictions. These tools can be used to monitor the performance and health of systems, but can also extract that information to be used as metrics for the ML models. The ones used to get the features that are planned to be used in the models developed are Prometheus, Node Exporter, cAdvisor, Telegraf, InfluxDB, FluentBit, and PyShark.

Prometheus is an open-source monitoring and alerting toolkit, which excels in its ability to collect, store, and query time-series data. It was designed with a focus on reliability and scalability, making it a great option for Cloud systems, as it tackles the scalability problem mentioned earlier and is particularly well-suited for dynamic cloud environments. Its flexible querying language, PromQL, allows users to easily manipulate the path of the collected metrics. It regularly scrapes metrics from the configured targets, enabling real-time monitoring and, therefore, extraction, facilitating the detection of performance bottlenecks or any other type of anomaly. It can be integrated with alerting rules to further empower administrators to respond proactively to any occurring issues, ensuring the stability and availability of the cloud service.

Node Exporter is an essential tool within the Prometheus ecosystem, designed to collect and expose hardware and OS metrics. It allows administrators to monitor various system metrics such as CPU usage, memory consumption, disk I/O, and network statistics. It is lightweight and highly extensible, supporting a wide range of metrics out of the box, while also allowing for custom metrics to be added through textfile collectors.

cAdvisor (Container Advisor) is a lightweight container monitoring tool developed by Google that specializes in providing real-time metrics about the resource usage and performance of containers. It can be integrated seamlessly with container orchestration platforms, such as Kubernetes, Docker, or Minikube making it a valuable tool for organizations that pretend to leverage containerized applications in the cloud. This tool can collect and expose metrics related to CPU, memory consumption, filesystem I/O, and network activity for individual containers, allowing administrators to optimize resource allocation, troubleshoot performance issues, etc. These values can also be exported to be used as metrics for ML models. The integration of cAdvisor and Prometheus can further enhance the monitoring ecosystem, allowing for a unified and comprehensive view of both virtualized and containerized resources.

Telegraf is a powerful and flexible open-source agent designed for collecting, processing, and sending metrics and events from various sources. Developed by InfluxData, Telegraf supports a wide array of input plugins to gather data from diverse systems, applications, and services, including databases, message queues, cloud providers, and more. It also features numerous output plugins to route collected metrics to different destinations, such as InfluxDB, Prometheus, and other time-series databases.

InfluxDB is a high-performance, open-source time series database, also developed by InfluxData, optimized for handling large volumes of time-stamped data, such as metrics, events, and logs. Designed to manage the time series data's unique characteristics efficiently, InfluxDB excels in storing, querying, and visualizing time series data with precision and speed.

Fluent Bit is a lightweight and efficient open-source log processor and forwarder, designed for collecting, processing, and shipping log data from various sources to different destinations. As part of the Fluentd ecosystem, Fluent Bit provides a streamlined and resource-efficient alternative, making it ideal for environments with limited resources, such as edge computing and IoT devices.

PyShark is a Python wrapper for the popular network protocol analyzer Wireshark, providing a powerful tool for capturing, parsing, and analyzing network traffic directly within Python scripts. By leveraging the capabilities of Tshark, Wireshark's terminal-based counterpart, PyShark allows developers and network engineers to automate network packet analysis and integrate it into larger workflows or applications.

In this system, cAdvisor and Node Exporter each run per host to collect container and host metrics, respectively, exposing them via a '/metrics' endpoint in Prometheus format. Prometheus and Telegraf periodically scrape these endpoints, with Prometheus storing the data and Telegraf sending it to InfluxDB after converting the metrics. An agent using PyShark runs on each host, capturing network packets continuously and storing them in InfluxDB. Fluent Bit collects and forwards container logs to InfluxDB. The admin interface manages active services and allows configuration of which metrics to monitor or drop, updating Prometheus and Telegraf configurations accordingly.

2.5 Summary

In this chapter, background information was presented. This information served as a foundation of knowledge that should help the reader understand more about the key topics addressed throughout this thesis, including cloud security, intrusion detection systems, machine learning, and feature extraction tools. The importance of cloud security is emphasized and challenges, strategies, technologies, and emerging trends are presented. The core functionalities of intrusion detection systems are also addressed, as well as some challenges and considerations that come with them. Finally, the feature extraction tools that were used are mentioned, presenting the diagram of the implemented solution.

Chapter 3

Related Work

This chapter presents the information regarding related work. First, the procedure followed to obtain the papers is described. After that, the information gathered is presented, such as the different types of cloud security areas addressed, the ML algorithms and datasets used, and how the approaches were evaluated. The goal of this section is to provide the reader with a set of examples of other works similar to this one.

3.1 Query Strategy

The search for documents was made in the digital libraries Google Scholar and IEEE Xplore, using the following keywords:

- Machine Learning
- ML
- Intrusion Detection System
- IDS
- Cloud
- Security
- Cloud Security
- Anomaly
- Network

These were used in combinations using Boolean logic (AND, OR, quotations, parenthesis) to get results more tailored to the expected. A systematic review that had the same objectives as this literature review was found, covering papers between the dates from 2007 to 2019. The analysis made by the researchers was

thorough and detailed, focusing on relevant information and detailing their process, however, some of the information presented in the tables shows a lack of knowledge in the area. Nevertheless, this doesn't invalidate their search, which will still be used to cover the dates previously mentioned. The data shown in the next tables will contain information extracted from this document but will be altered to correct the previously mentioned mistakes.

3.2 Cloud Security Areas

In the 63 papers collected "Machine Learning for Cloud Security: A Systematic Review" [Nassif et al., 2021], together with the 10 collected from the previously described strategy, 10 cloud security areas are addressed and researched. Table 1 shows the number of research papers in each cloud security area and the percentage that shows the frequency of that area.

Table 1: Cloud security areas addressed

Cloud Security Areas	Frequency	Percentage	Description
Threat Detection	29	40%	Continuous monitoring to identify and respond to potential security breaches or malicious activities in cloud environments
DoS & DDoS	14	19%	Defending against Denial of Service (DoS) and DDoS attacks to ensure uninterrupted cloud service availability
Data Privacy	9	12%	Safeguarding sensitive information, ensuring compliance with privacy regulations and protecting user confidentiality
Malware	7	10%	Preventing, detecting, and/or removing malicious software to secure cloud systems and data from unauthorized access or damage
Privacy Preservation	6	8%	Uphold individual privacy rights, restricting access to personal data, and ensuring responsible data handling
Security	5	7%	Comprehensive measures to protect cloud infrastructure, applications, and data from various threats
Confidentiality of Data	3	4%	Ensuring that sensitive information remains private and secure, employing encryption and access controls

Although all of the topics are relevant, the ones that closely relate to this thesis are the ones concerning Threat Detection and DoS & DDoS. Contained in Threat Detection, two more specific categories are relevant for this thesis: Anomaly Detection and Intrusion Detection. Detection of anomalies involves finding deviations from normal anticipated behavior through the analyses of patterns. A total of 6 papers covering anomaly detection were found by the authors of the systematic review, with 3 of them concerning user behavior rather than system behavior. One more was found from the research for the thesis. For the sake of easily identifying each paper, a table can be found in Appendix A addressing an ID to each

one. Table 2 summarizes the contents of each paper that concerns anomaly detection:

Table 2: Summary of papers regarding anomaly detection

ID	Paper summary
P1	Proposes an anomaly detection system at the hypervisor layer named Hypervisor Detector that uses a hybrid algorithm which is a mixture of Fuzzy C-Means clustering algorithm and Artificial Neural Network (FCM-ANN) to improve the accuracy of the detection system [Pandeeswari and Kumar, 2016]. It shows great results for attacks with low frequency, however, when compared to the other models, it shows the same or worse performance than simpler models, like Naive Bayes.
P2	Investigates both detecting and categorizing anomalies rather than just detecting, which is a common trend in contemporary research works, using two supervised machine learning techniques, namely linear regression (LR) and random forest (RF). Although the detection is very good, the categorization can be less accurate due to similarities between attacks [Salman et al., 2017].
P3	Proposes a new SVM approach, named Enhanced SVM, which combines soft-margin SVM and one-class SVM to provide unsupervised learning and low false alarm capability, similar to that of a supervised SVM approach [Shon and Moon, 2007]. The overall goal of this approach was reached, however, it was not tested in real-world TCP/IP traffic.
P4	Proposes a framework to utilize anomaly detection and random re-sampling techniques for profiling user's behaviors via the frequent patterns of activated system processes [Chiu et al., 2013]. The results showed an average detection of 86%, however, the approach is not very clear about their methodology.
P5	Presents a new mobile cloud infrastructure that combines mobile devices and cloud services and proposes a methodology and architecture for detecting abnormal behavior through the monitoring of both host and network data [Kim et al., 2012]. The methodology and experimentation are very detailed and the testing is thorough.
P6	Proposes a new QoS that is capable of monitoring and detecting insider threats using KNN. Also proposes a detection model for insider threats, which utilizes Facial recognition and Monitoring models [Sarma et al., 2017]. Although the system is smart, using facial recognition might be an unnecessary privacy invasion, so this would not be recommended for broad applications.
P70	Proposes an efficient Hybrid clustering and classification models for implementing an anomaly-based IDS for malicious attack type classifications such as DoS, Probe, U2R, and R2L using threshold-based functions [Samunnisa et al., 2023]. It presents very detailed results and an in-depth methodology.

IDSs can be used to defend systems from different kinds of attacks and is the main focus of this thesis. Nine of the selected papers for the systematic review take up intrusion detection for cloud systems, while 7 other ones were then added to the list. Summaries of the content of the collected papers can be found in Table 3. The ones obtained from the systematic review have been concealed as the table would be too big.

Table 3: Summary of papers regarding intrusion detection

ID	Paper summary
P65	Proposes a transfer learning IDS based on the Convolutional Neural Network (CNN) architecture that has shown excellent results on image classification, using five pre-trained CNN models, including VGG16, VGG19, Inception, MobileNet, and EfficientNets, to train on two selected datasets: CIC-IDS2017 and CSE-CICIDS2018 [Okey et al., 2023b]. It shows great results, as all models show results between 97 and 100% for Accuracy, Precision, Recall, and F-score.
P67	Presents a cloud-based intrusion detection model based on random forest (RF) and feature engineering. Specifically, the RF classifier is obtained and integrated to enhance the accuracy (ACC) of the proposed detection model [Attou et al., 2023a]. The model shows great performance, however, recall is far from ideal when tested on the NSL-KDD dataset.
P68	Proposes the methodology of an intrusion detection scheme utilizing a deep learning technique which is based on Fuzzy Min-Max Neural Networks-Based Intrusion Detection System (FMMNN-IDS). It also investigates the model's performance in terms of binary and multiclass classification, as well as the count of learning rate and neurons that affect the model's performance [Kumar et al., 2022]. The results shown only cover accuracy and detection rate, which, although present good results, can be insufficient for certain scenarios.
P69	Proposes a logistic regression-based oppositional tunicate fuzzy C-mean (LR-OTSFCM) methodology for the detection of cloud intrusion, with a focus on attack detection from a cloud environment [Kanimozhi and Aruldoss Albert Victoire, 2022]. The model is well tested and is compared with state-of-the-art approaches demonstrating a superior attack detection rate using four datasets.
P71	Provides an effective optimal security solution for intrusion detection in a cloud computing environment using a hybrid deep learning algorithm (EOS-IDS). Unnecessary data is removed with improved heap optimization (IHO) and feature selection is done using chaotic red deer optimization (CRDO) [Mayuranathan et al., 2022]. Although the proposed model provides good values for certain metrics, it still gets outperformed in others, and the results are not very clear in that sense.
P72	Proposes a new fuzzy deep neural network (FDNN) with Honey Bader Algorithm (HBA) for privacy-preserving intrusion detection technique, named FDNN-HBAID for cloud environment, based on the design of an intrusion detection approach with a blockchain-enabled privacy-preserving scheme [Jain et al., 2023]. The results presented show that the model accomplishes minimal values of training and validation losses while achieving a performance close to 100% accuracy and f1-score. More metric values would be useful, for an even deeper understanding of the model's performance.
P73	Presents two models based on deep neural networks (DNNs) for this study: the first model is built on a multi-layer perceptron (MLP) with backpropagation (BP), and the other is trained by MLP with particle swarm optimization (PSO) [Alzughaibi and El Khediri, 2023]. The results are well documented and present various metrics that show the performance of the proposed models is on par with the ones mentioned in the related works. The authors also provide some essential tips that they think are useful for researchers in the field.

DoS attacks prevent lawful customers from accessing the network or other resources. A DDoS is a type of DoS attack, in which the attacker increases the number of sources to similarly increase the output of requests, resulting in even bigger outputs of information that systems cannot handle. From the papers collected in the systematic review, 13 enter this category, with 3 of them addressing DoS in general and the other 10 focusing on DDoS. 1 other paper was then found that also addressed this area, bringing the total to 14. Table 4 shows summaries of

some of the papers from the systematic review and the one found for this thesis.

Table 4: Summary of papers regarding DoS & DDoS

ID	Paper summary
P12	Presents a statistical technique to detect and filter DDOS attacks, with the requirements of having small storage and the ability of fast detection [Shamsolmoali and Zareapoor, 2014]. The results show great performance when compared to state-of-the-art and DDoS detection models, however, only accuracy, false alarm rate, and detection time are used as metrics, which might be insufficient to guarantee the utility of the model in all cases.
P15	Proposes a DoS attack detection system on the source side in the cloud, based on machine learning techniques, which leverages statistical information from both the cloud server's hypervisor and the virtual machines, to prevent network packages from being sent out to the outside network [He et al., 2017]. The results show that DoS attacks are successfully detected and the approach does not degrade the performance of the system. It also uses various metrics to analyze the model, giving a very in-depth and detailed performance overview.
P27	Proposes an approach to mitigate insider attacks using a host-based user profiling technique where a keystroke dynamics is used for analyzing the user behavior and a retraining approach is also proposed as the imposter patterns are absent at the time of registration [Bondada and Bhanu, 2014]. Although the approach can detect insider attacks, using a key logger to monitor users is very privacy-invasive, which would not fit well on a public system.
P32	Offers a solution to traceback through Cloud TraceBack (CTB) to find the source of some of the current attacks that attackers may initiate as HTTP and XML, and introduces the use of a back propagation neural network, called Cloud Protector, which was trained to detect and filter such attack traffic [Chonka et al., 2011]. The paper presents a very thorough explanation of two lethal attacks and the solution presents valid results, however, these are not very clear.
P35	Uses extreme gradient boosting (XGBoost) as a detection method in SDN-based cloud and simulates real DDoS attack environments [Chen et al., 2018]. The algorithm can detect DDoS with higher accuracy and lower false positive rate than other popular algorithms, and is extraordinarily quick, being able to adapt to a high-speed environment. It is also scalable, which is very important for the cloud due to its continuously expanding network.
P48	Proposes techniques to secure a cloud environment by incorporating some of the efficient approaches in intrusion detection, focusing on two major issues in IDS: efficient detection mechanism and speed of detection [Mishra et al., 2016]. The proposed techniques present great results, however the performance on U2R attacks should be increased, as the detection of those falls behind the rest of the other attacks.
P66	Proposes a hybrid algorithm consisting of several machine learning techniques combined to detect and classify the type of DDoS attack with greater accuracy than that of each model [Varma et al., 2022]. The document is rather small but can fit some information into it, however, only accuracy is used to measure the performance of the models, which is rather superficial.

3.3 Machine Learning Algorithms

In the gathered documents, many ML algorithms were applied by researchers to their approaches. The list of the 20 algorithms can be seen in Tables 6 and 5:

Table 5: Classical Machine Learning algorithms used in the papers

ML Models	Paper IDs	Training Type
SVM	P3, P8, P11, P16, P18, P25, P28, P34, P35, P37, P38, P39, P40, P41, P42, P48, P54, P55, P64, P66, P70	Supervised
Random Forest	P2, P5, P9, P13, P40, P56, P58, P59, P64, P66, P67, P70	Supervised
KNN	P6, P35, P37, P46, P47, P48, P61, P62, P64, P66, P70	Supervised
Decision Tree	P26, P43, P44, P49, P51, P53	Supervised
Naive Bayes	P59, P50, P64, P66	Supervised
K-means	P17, P56, P70	Unsupervised
Linear Regression	P2, P13	Supervised
Linear Means Classifier	P7, P22	Supervised
Logistic Regression	P29, P60	Supervised
Fuzzy C-means	P38, P69	Supervised/Unsupervised
FLD CLassifier	P7	Supervised
Bayes Net	P10	Supervised/Unsupervised
Polynomial Regression	P21	Supervised
CKNN	P52	Supervised
XGBoost Classifier	P36	Supervised
Gradient Boosted Classifier	P64	Supervised
GMM	P70	Unsupervised

Table 6: Deep Learning algorithms used in the papers

ML Models	Paper IDs	Training Type
Neural Network	P1, P19, P23, P24, P30, P31, P32, P33, P45, P46, P49, P53, P55, P57, P64, P65, P68, P71, P72, 73	Supervised/Unsupervised
LSTM	P57, P71	Supervised
C_2DF	P12	Supervised

The most used algorithm was SVM, with some variants like LS-SVM and One-class SVN also being used as standalone models. Neural Networks follow closely, with 20 entries, with multiple variants across papers. Random Forest was used 12 times, with five of those using it as a standalone model, followed by KNN, which was used as a standalone model in 6 out of 11 studies. Decision Trees were also popular, used in 5 studies, with only one not using it as a standalone solution. Naive Bayes came close, being used in 4 different papers. Every other model was only featured in a smaller number of papers, however, some papers combined more than one model to create hybrid solutions with, once again, SVM being the most common one. Table 7 shows the hybrid models used in the papers:

Table 7: Papers that used hybrid models

ID	Hybrid Model
P2	Linear Regression + Random Forest
P11	SVM + Linear SVM
P13	Random Forest + Linear Regression
P37	KNN + SVM
P38	SVM + Fuzzy C-Means
P40	SVM + Random Forest
P46	ANN + KNN
P49	Neural Network + Naive Bayes + Decision Tree
P54	SVM + Neural Network
P57	CNN + LSTM
P66	SVM + KNN + Naive Bayes + Random Forest [Varma et al., 2022]
P70	K-Means + Random Forest & GMM + Random Forest & KNN + SVM [Samunnisa et al., 2023]
P71	CNN + LSTM

3.4 Datasets

Not all papers addressed implemented models, however, those that did, required a dataset to train those models. From the systematic review, 12 concrete datasets were mentioned across 34 papers, with 2 other datasets being addressed in the other documents collected. There were also datasets that were synthetically created specifically for some papers as well as datasets that were not mentioned by name or non-public. Table 8 represents the datasets used, along with how many times they were mentioned across the papers.

Table 8: Datasets used in collected papers

Dataset	Mentions	Size
UNM	1	?
Kyoto 2006+	1	93591959 lines, 24 columns
NSL-KDD	7	149470 lines, 41 columns
ANTIY Laboratory	1	?
KDDCUP 99	5	4898431 lines, 41 columns
MNIST	1	70000 images
NUS-WIDE	1	269648 images
CAIDA	2	Requires request
Real-Life Data	6	Varies
MalGenome	1	3800 lines, 216 columns
Wisconsin Breast Cancer Dataset	1	570 lines, 32 columns
UNSW	3	2540044 lines across 4 files, 49 columns
DARPA	2	?
KDD	5	Varies
CIC-IDS2017	4	2830751 lines across 8 files, 79 columns
CSE-CICIDS2018	3	16232999 lines across 10 files, 84 columns
Non-public or non-specific datasets	7	Varies

The most used datasets are from KDD. These can be NSL-KDD with 7 mentions, or KDDCUP 99 with 5. "KDD" was only mentioned in many papers, although that is not a concrete dataset, so it should fall into one of the two mentioned. The Canadian Institute of Cybersecurity datasets were also popular, these being CIC-IDS2017 and CSE-CICIDS2018. UNSW was mentioned in 3 papers each and CA-DIA and DARPA were used in 2. All of the other datasets were only mentioned once, including the ones that fit into the "Non-public or non-specific" class.

The Kyoto 2006+ dataset was built on three years of traffic data from November 2006 to August 2009. It consists of 14 features that were derived from the KDD-CUP 99, as well as 10 additional ones. It was captured using honeypots, darknet sensors, an email server, and a web crawler. These mechanisms were deployed on the five networks inside and outside the Kyoto University and collected all traffic data. There were around fifty 50 normal sessions and 43.5 million attack sessions, with around 500 thousand of those being related to unknown attacks. As mentioned, the dataset was based on the KDDCUP 99, but the authors excluded substantially redundant and insignificant features, as they were not suitable for network-based IDSs, the focus of this dataset.

The NSL-KDD dataset was also based on KDDCUP 99, and it was developed to fix issues in that dataset, such as the high number of redundant records (78%) and duplicate records (75%), which prevented the proper classification of the other records. This dataset consists of a reasonable number of selected features from the original dataset, containing 21 out of the 37 attacks present in the KDDCUP 99. The normal traffic consists of around 126 thousand instances, out of which around 77 thousand consist of anomalies.

The KDDCUP 99 is a staple of anomaly detection in computer networks. It consists of a collection of data transfers from virtual environments that was used on the Competition of the Third Knowledge Discovery and Data Mining Tools in 1999. It is a subset of DARPA, a dataset that was collected by simulating the operation of a typical US Air Force LAN, by the MIT Lincoln Laboratory. It consists of 4898431 lines, each having 41 features.

The CIC-IDS2017 dataset was created by profiling the abstract human interactions and generating naturalistic benign background traffic, based on the HTTP, HTTPS, FTP, SSH, and email protocols of 25 users. It was captured across 5 days and includes attacks such as Brute Force FTP, Brute Force SSH, DoS, etc. It consists of almost 3 million lines across 8 files, each having 79 features.

CSE-CICIDS2018 is a dataset created in a collaborative project between the Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC). With 79 features and around 1.5 million lines across 10 files, this dataset features seven different attack scenarios, present on the captured traffic of real labeled network traffic.

3.5 Model Evaluation and Analysis

From the systematic review, 36 papers showed performance metrics in their research, with the 10 others found in the research also mentioning them. Across all of the papers, there are over 30 metrics, with some only being mentioned once, therefore, those will be discarded, as they are very niche and were only used in specific situations. Table 9 shows the number of times each metric was mentioned:

Table 9: Metrics used in collected papers [Nassif et al., 2021] [Kimmell et al., 2021] [Varma et al., 2022] [Okey et al., 2023a] [Attou et al., 2023b]

Metric	**Mentions**
True Positive Rate	27
Accuracy	24
False Positive Rate	18
Precision	15
F1 Score	9
True Negative Rate	7
Detection	6
False Negative Rate	2
Training Time	2

The most used metric was True Positive Rate, also known as Recall or Sensitivity. It measures the ability of the model to obtain all positive instances in the dataset successfully, It calculates the ratio of correctly predicted positives to the total number of actual positive instances. Recall is crucial when the cost of false negatives is high ensuring that the model identifies as many positive instances as possible. Accuracy was close, with 24 papers using it, which is a metric that measures the overall correctness of the model. It calculates the ratio of correctly predicted results to the total number of results. While it provides a general overview of the model's performance, accuracy may not be suitable for some datasets, such as ones with imbalanced data, where one class significantly outnumbers the others. False Positive Rate (or False Alarm Rate) is used in 18 papers, which is the probability of falsely classifying a non-positive event as a positive one. 15 papers used Precision, a metric that assesses the accuracy of positive predictions made by the model, using the ratio of correctly predicted positives to the total number of positive predictions. This value is particularly valuable when the cost of false positives is high and helps understand how reliable the positive predictions are. F1 Score, which is a composite metric that combines precision and recall into a single value was used 9 times. This allows us to easily check the balance between precision and recall, making it particularly useful in scenarios with imbalanced class distribution. True Negative Rate was used in 7 papers to represent the chance of correctly identifying non-positive events as such. Detection was used 6 times and the last two, False Negative Rate and Training Time, were used only 2 times each.

This shows a preference for some metrics, particularly Accuracy, Precision, and

Recall, which are frequently used together in multiple papers. This set of metrics provides a very broad sense of how the models are performing, covering many areas that are more or less important, depending on the purpose of the models.

3.6 Summary

This chapter provided an overview of the literature review process, outlining the search strategy that was used to collect relevant papers on the topic. After this, the different cloud security areas addressed in the papers are presented, focusing on the ones that are relevant to this thesis. The various ML algorithms used were then presented, showing trends in the use of certain models, as well as the hybrid solutions that were also implemented. The datasets that were used to train these models were also shown, highlighting the popularity of the KDD datasets. Finally, the mc common evaluation metrics employed by researchers were addressed.

Chapter 4

Approach and Methodology

In this chapter, the topics that are addressed are related to how the work was done. The first section presents the expected outcome and the proposed approach to reach that outcome. After that, a section related to the methodologies and mechanisms follows, touching on topics such as the selection of the model and dataset, the evaluation of features and how that influenced which ones were chosen, and finally how the models were evaluated in each of the two used methods and the version of the tools and frameworks that were used.

4.1 Expected Outcome

At the end of this thesis, it is expected to have a set of models that, in conjunction with feature extraction tools, can be trained to detect anomalies in cloud systems. The goal is for these models to provide the best possible results with the smallest possible set of features, not only to reduce the size of the datasets and the time it takes to train the models but also to reduce the amount of information that is being tracked. The models should be easily deployable, together with the tools that extract the data needed to feed them. To achieve this, they should be integrated into the deployable product that is being developed for project NEXUS. The work done for this project can be checked on NEXUS GitLab, although the access must be granted on request, as it is currently closed access and requires a VPN.

4.2 Proposed Approach

For the execution of this work, the following will be done:

1. **Select Machine Learning models:** The models that will be used to classify the anomalies will be selected.
2. **Select the dataset to train the models:** The dataset that will be fed to the models will be selected.

3. **Implement models and dataset:** Both the models and dataset will be put together and the first tests will be executed.

4. **Validation:** The results of the implementation will be validated, checking, for example, the confusion matrix, to see if the models are not always predicting the same thing.

5. **Evaluation:** The effectiveness of the models will be evaluated, using metrics like accuracy and precision. The time taken to train and predict will also be taken into account.

6. **Feature Engineering:** Modifications to the dataset will be done, reducing the number of features used and trying combinations of some of them to improve the performance of the models.

7. **Refine approach:** Any modifications needed to accommodate for any suggestions and conclusions will be made.

4.3 Methodology

In this section, the methodology used will be discussed. Starting with the ML models that were selected as well as the selection of the dataset that was used to train and evaluate those models. After that, the methods used to rank the importance of the features will be described, as well as how those evaluations lead to the metric selections, reducing the dataset size. After that, there will be a subsection that will touch upon the evaluation of the models, describing how the dataset was used to train and test the models in both methods. After that, there is a section that touches on the metrics used to measure the performance of the models, and the last section focuses on the tools and frameworks that were used to implement all of the mentioned work. By the end of this section, the reader should have a clear understanding of the methods and mechanisms employed during the practical part of this thesis.

4.3.1 Model and Dataset Selection

The model selection began during the literature review and state-of-the-art familiarization part of the plan. During this time, many documents were collected and read with the main goal of understanding what models were currently being used for cloud environments. Using Tables 6 and 5, it was possible to see trends in certain models, so that was taken into account when choosing them. Another piece of information analyzed in the literature review was the datasets used in the collected papers, another useful piece of data. Table 8 showed that KDD datasets and real-life datasets were the most popular ones. A second and third datasets were later added by recommendation of the jury of the intermediate delivery of this thesis.

4.3.2 Feature Evaluation and Selection

For the feature evaluation, various methods were used, some relying on information extracted from the models, and others using statistical information from the dataset. One was to check the coefficients provided by the models. This gives us a table in which for every (feature, class) pair, a value is given. This value is used to calculate the probability of a data point belonging to each class in the classification model. This means that, theoretically, the higher values a column has, the more relevance it has, as it affects the decision of the model in a greater way than one with smaller values.

Another method that was based on the information provided by the models was Random Forest's 'feature_importance_' function, which directly provides impurity-based feature importances [Pedregosa et al., 2011]. This means that the importance of each feature is given by how much it helps to reduce impurity and accurate predictions [Youssefi, 2023]. This method provides a more clear result on the feature's importance, giving a percentage to each one.

For the statistical-based methods, one of them was Fisher Score, a feature selection approach that ranks features based on their ability to differentiate various classes in the dataset [Rosidi, 2023]. One difference from this method to the previous ones is that it only needs to be run once, due to the nature of variance, which is what Fisher Score uses to rank the features. When removing features, the variance of the other ones will either remain the same or go up.

For the final method, instead of trying to figure out the importance of each feature, it aims to identify the features of the dataset that follow the same pattern, and, therefore, there should be no statistical impact if all but one of them gets removed. To achieve this, two functions were used. The first one is the Pearson Correlation, which measures the linear correlation between two sets of data, while the second one is Spearman's Rank Correlation, a variation of the Pearson Correlation that assesses monotonic relationships between two sets of data. These functions output a value between -1 and 1, and the closer it is to one of the two ends of the spectrum, the bigger the correlation between the two features.

Having these 4 methods to use, the feature engineering process began. Based on the outputs of the 4 different methods, features were removed from the datasets, resulting in different results.

4.3.3 Evaluation Approach

This process had two methods, one that was used before the intermediate delivery of this thesis, and another one that was developed keeping in mind the changes suggested by the jury and the advisors. To better identify these solutions, the first one will be named Fisher-RF, and the second one will be named Cross-Validated CoRF. The results of these two solutions will also be split in this thesis. The results obtained with the first version (which was presented in the intermediate delivery) will be presented in Chapter 5, while the results obtained with the second and final version will be presented in Chapter 6.

4.3.3.1 Fisher-RF

For the evaluation approach of this method, the dataset is divided into two with a split of 80/20. The smallest split is then labeled as Testing Data, which is used to make the final evaluation of each model. The biggest split is passed through a scaler and labeled as Training Data. This data is then used to train the first instances of the 3 standalone models: Support Vector Machine (SVM), Gaussian Naïve Bayes (GNB), and K Nearest Neighbour (KNN). Additionally, it is also passed through a Blending Ensembling Technique, being divided into an 80/20 split once again, as it will be used to both train and test three separate instances of the previous models. The predictions of those models are then added as features to the dataset and fed to a hybrid model that uses Random Forest. The final 4 models (the 3 initial standalone ones and the hybrid one) are then evaluated using the smallest split that was taken away at the beginning of the process to prevent any possible previous knowledge. This process can be seen as a diagram in Figure 5.

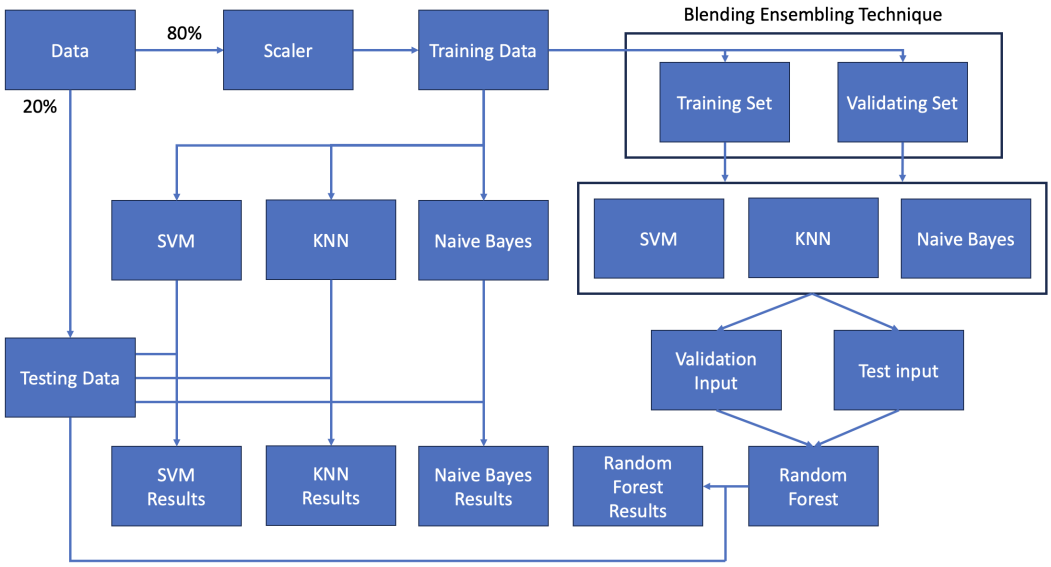


Figure 5: Evaluation process

For this method, three of the four solutions presented for the feature removal process were tested: Fisher Score, Model Coefficients, and Random Forest Feature Importance. Out of these three, Fisher Score and Random Forest Feature Importance were implemented, while Model Coefficients were not, as it was not ready to implement at the time of the intermediate delivery due to a time constraint. The goal of the removal of the features was to make the models faster while keeping their performance, or even improving their predictions, with the removal of possible deteriorating features.

4.3.3.2 Cross-Validated CoRF

For this method, the hybrid model was dropped due to an incompatibility of the previous code with the important introduction of cross-validation using K-fold, however, the Random Forest model was still kept in, but running as a standalone model. SVM, however, was completely ditched, due to its poor scaling, that was considerably impacting the speed of the tests. To compensate for this lack of scaling, parallellism was considered, however, no working solution was found for SVM. This was not the case for KNN and Random Forest, with both simply requiring the parameter 'n_jobs' to be set to the number of CPU cores to be used or to -1, to use the max amount of available cores.

A new column was also added to the first dataset to match the labeled columns of the second and third ones. One that identified the specific attack, and another one that worked binarily, with a 1 identifying an attack and a 0 meaning it was normal network traffic, to further improve the performance of the models, as it is easier to distinguish between two states than it is to distinguish between, in this case, five.

Regarding the feature selection methods, following the results and observations previously made in the first method regarding Fisher Score, this method was dropped and replaced by the correlation solution. As for the Coefficients approach, it was implemented using the coefficients generated by the GNB model, as it was the only of the three used models with this function available.

In terms of dataset balancing, the first method only used a limitation approach (by removing lines from the dataset in order to balance everything by the smallest sample of labels. This, although valid, could heavily reduce the amount of data in the datasets, especially when the attacks are identified in a non-binary way. To combat this, by the suggestion of one of the members of the jury, another balancing option was added, using Smote, a technique that generates new synthetic samples by interpolating between existing minority-class samples. Figure 6 shows the diagram of the new method.

When the data is imported, it first goes through the Pearson Correlation and Spearman Correlation functions without removing any feature. It then gets prepared to be fed to the models, by removing any unwanted columns, balancing the data, and separating the column that will be used for labeling. This data preparation is all controlled by the parameters shown in the purple box, which can be easily changed to modify how the data is handled, making it possible to easily switch between datasets, feature removal, data balancing, and binary or non-binary attack identification.

The first parameter, 'Dataset', indicates which of the three datasets to use. If it is set to 'DDoS', the dataset used in the first method will be used, if it is set to 'EdgeIIoT-DNN', the bigger version of the EdgeIIoT dataset will be used, the one meant to be used with Deep Neural Networks. When set to 'EdgeIIoT-ML', the dataset used will be the smaller version of the EdgeIIoT dataset and, finally, when set to 'UCDataset' it will use the dataset that was provided by the University of Coimbra.

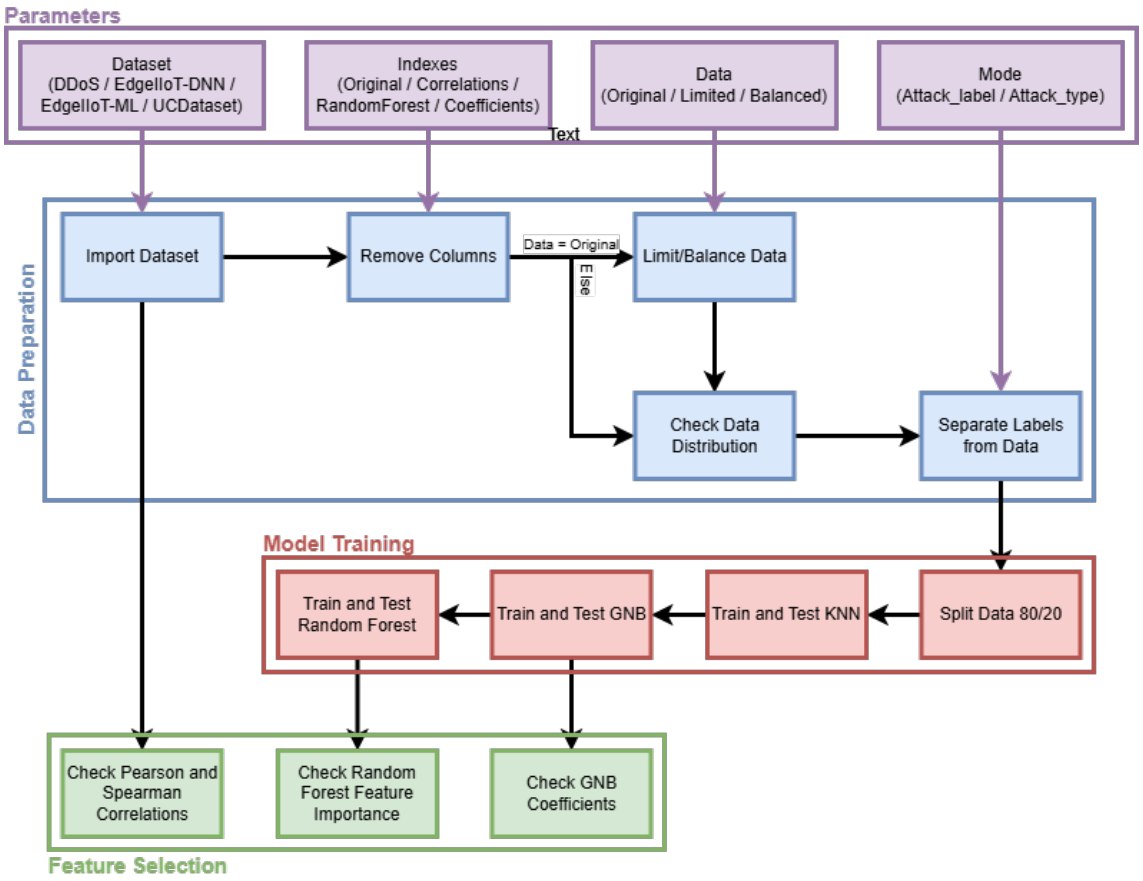


Figure 6: Diagram of the second method used

The second parameter, 'Indexes', is responsible for the choice of what columns get removed from the dataset. These are identified in lists that have been manually populated by training the models with the Original indexes and checking the Pearson and Spearman Correlations, Random Forest Feature Importance, and the GNB Coefficients, which correspond to the values 'Correlations', 'RandomForest', and 'Coefficients', respectively.

The third parameter, 'Data' controls how the balancing of the data is done. If this parameter is set to 'Original' no balancing is done. Otherwise, if set to 'Limited', the balancing that was used in the first method is employed, by limiting the number of instances of each label to the smallest one of them. As mentioned before, this can have a great hit on the performance of the models, especially when used along with non-binary labeling. This is why the third option, 'Balanced', was introduced. This one uses Smote to populate the dataset with new synthetic values that balance it by adding instead of taking away from it.

The last parameter, 'Mode', defines which of the two columns that identify the attacks is used. If set to 'Attack_label', the binary column will be used, while if set to 'Attack_type', the column that specifically identifies what attack is being made will be used. Due to time restrictions, the latter was not included in the result analysis, which was not a big loss because the performance of the models was very weak when compared to the binary solution.

After the data gets prepared, it then goes to the model training part of the code, being fed to each model at a time. Figure 7 shows how this portion of the code trains each model.

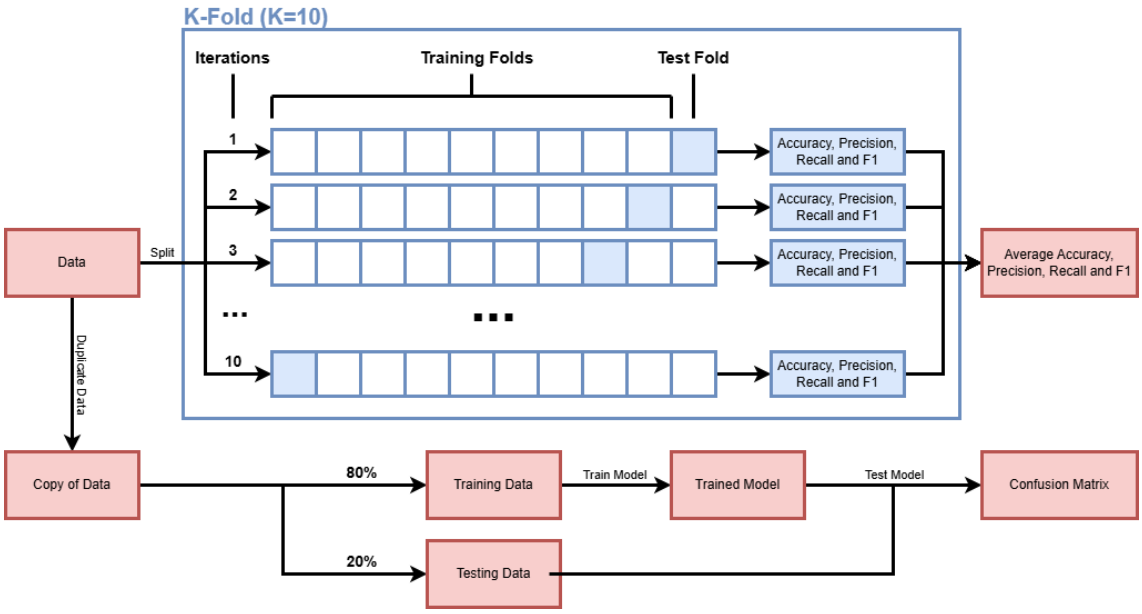


Figure 7: Diagram of the model training portion of the second method

First, the data gets duplicated and passed through a basic train-test split of 80/20. This is done because, the way the K-fold is implemented, there is no access to the predictions of the models, therefore the confusion matrix can't be calculated. After that, the original data is sent to a K-fold where, in this case, the K is equal to 10.

This means that, as seen in Figure 7, the data gets split into 10 parts, out of which one is set as the testing data, and the rest is set as training data. The model is trained and tested, resulting in accuracy, precision, recall, and f1 scores. The part that was used as testing data, goes back to being training data, while one of the other ones takes its place, and the testing and training is re-done. The process is repeated until all of the 10 parts have been used as testing data, and then the average of the accuracy, precision, recall, and f1 scores is the result of the function.

This method is important because it provides a more reliable estimate of the models' performance compared to a single train-test split, ensuring that the models are consistent across different subsets of the data.

4.3.4 Evaluation Metrics

To get the prediction results, Accuracy, Precision, Recall, and F1 Score were the main metrics used. These are calculated using the number of True Positives, True Negatives, False Positives, and False Negatives, and can give us a better understanding of how the model is performing. Figure 8 shows how each of those metrics is calculated.

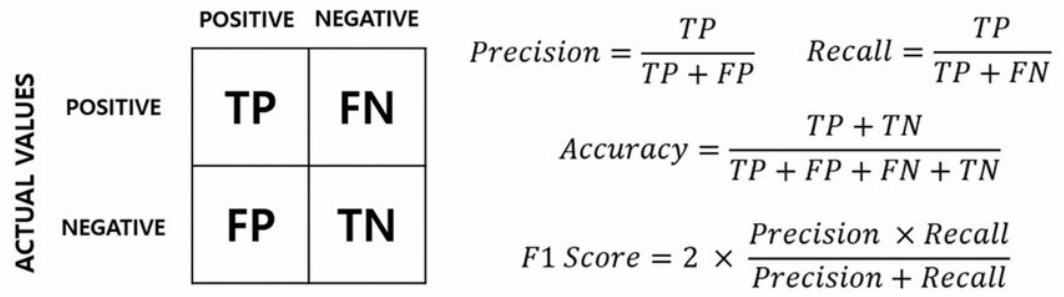


Figure 8: Evaluation metrics used [Seol et al., 2023]

Accuracy is a metric that measures the overall correctness of the model. It calculates the ratio of correctly predicted results to the total number of results. While it provides a general overview of the model's performance, accuracy may not be suitable for some datasets, such as ones with imbalanced data, where one class significantly outnumbers the others.

Precision assesses the accuracy of positive predictions made by the model, using the ratio of correctly predicted positives to the total number of positive predictions. This value is particularly valuable when the cost of false positives is high and helps understand how reliable the positive predictions are.

Recall (also known as True Positive Rate, measures the ability of the model to obtain all positive instances in the dataset successfully, It calculates the ratio of correctly predicted positives to the total number of actual positive instances. Recall is crucial when the cost of false negatives is high ensuring that the model identifies as many positive instances as possible.

F1 Score is a composite metric, calculated by combining precision and recall into a single value. This allows us to easily check the balance between precision and recall, making it particularly useful in scenarios with imbalanced class distribution.

Using these metrics, we got a sense of how the algorithm would perform in the real world, together with the training time and prediction time.

4.3.5 Tools and Frameworks

The experimental work was implemented on a Windows 11 machine running Python 3.11.5 on Jupyter Notebook 6.5.4.

The Pandas library, together with Numpy, was used to manage and manipulate the data. Liac-arff was used to import the data into the environment, as it was provided in the '.arff' format. Matplotlib, Seaborn, and IPython were used to better represent tables and matrixes, providing more control over how they were displayed. The Scikit-learn library was the most important one, as it provides all of the functions that are related to the models, as well as the models themselves.

All of the tools and framework versions can be seen in Table 10:

Table 10: Tools and Framework versions

Tool / Framework	**Version**
Python	3.11.5
Jupyter Notebook	6.5.4
Pandas	2.0.3
Numpy	1.24.3
Liac-arff	2.5.0
Matplotlib	3.7.2
Scikit-learn	1.3.0
IPython	8.15.0
Seaborn	0.12.2

4.4 Summary

In this chapter, the approach to tackle the objective of this thesis was presented. The expected outcome of the thesis was presented as well as the plan to reach it. The methodology was also addressed, regarding the selection of the dataset and the model, which were done based on information gathered in the literature review and the suggestions of the jury in the intermediate delivery. The feature evaluation and selection were also touched upon, addressing the four feature selection methods that are implemented in this paper. The evaluation of the models was also explained, demonstrating how the datasets are handled in each of the two methods presented. Finally, the functions of the tools and frameworks used were addressed, as well as their versions running on both the machines that were used during the process.

Chapter 5

Fisher-RF Results and Discussion

In this chapter, the results of the work done during the first half of the thesis will be presented and discussed. The practical implications of said results will also be addressed, as well as the limitations of the methodology.

5.1 ML Models and Dataset

The selection of models was made based on 6, in which a high usage of SVM, Random Forest, and KNN can be seen among classical ML models. Neural Networks, the most popular choice for Deep Learning, were also a valid choice, however, due to the low experience in the area and time restriction introduced by the thesis delivery, a decision was made to stick with the classical models.

With the model selection done, the search for an implementation of these began. One of the papers found, titled "Detection of DDOS attacks using machine learning techniques: A hybrid approach", by D Anurag Varma, Ravi Ashish, and V Sandeep, made an implementation of these exact models with the addition of GNB that showed promising results. Although the document is small, there is a video on Youtube titled "Detecting DDoS Attacks by ML Technique: Hybrid Approach | KNN SVM Gaussian Naïve Bayes | Source Code" that follows the same approach to implement these models, which served as a guide for the implementation of the first method used in this thesis.

The dataset used in both the paper and the video ended up being the first one used for this thesis as well, as it was network-focused and, during the initial week of the internship, it had been decided that this would be my main focus. This dataset is "DDOS Attack Network Logs", by Jacob van Steyn, and is available on Kaggle. It is from 2019 and contains around 2,100,000 labeled network logs from various types of network attacks: UDP-Flood, Smurf, SIDDOS, and HTTP-FLOOD. After some research, most of the features also looked like they were relatively simple to extract from a cloud environment.

5.2 Implementation and Validation

The initial implementation, as mentioned, was done with the help of a YouTube video [Ove, 2021] that used the same models and dataset as the ones chosen. The first implementations, shown in Figure 9, showed that the hybrid model did not have a better performance than all the non-hybrid ones, which went against the video and the paper on which the implementation was based.

```
[11:06:39] Started importing data... Finished! Time elapsed: 18s
[11:06:58] Started breaking data... Finished! Time elapsed: 2s
[11:07:00] Started limiting data... Finished! Time elapsed: 0s
[11:07:00] Started splitting data... Finished! Time elapsed: 0s
[11:07:00] Started scaling data... Finished! Time elapsed: 0s
[11:07:01] Started training SVM... Finished! Time elapsed: 76s
[11:08:17] Started predicting... Finished! Time elapsed: 5s. Accuracy: 89.464%
[11:08:23] Started training KNN... Finished! Time elapsed: 0s
[11:08:23] Started predicting... Finished! Time elapsed: 0s. Accuracy: 98.976%
[11:08:23] Started training GNB... Finished! Time elapsed: 0s
[11:08:23] Started predicting... Finished! Time elapsed: 0s. Accuracy: 90.408%
[11:08:23] Started splitting data... Finished! Time elapsed: 0s
[11:08:23] Started training SVM... Finished! Time elapsed: 73s
[11:09:37] Started predicting... Finished! Time elapsed: 8s
[11:09:45] Started training KNN... Finished! Time elapsed: 0s
[11:09:45] Started predicting... Finished! Time elapsed: 0s
[11:09:46] Started training GNB... Finished! Time elapsed: 0s
[11:09:46] Started predicting... Finished! Time elapsed: 0s
[11:09:46] Started training Random Forest... Finished! Time elapsed: 0s. Accuracy: 98.724%
```

Figure 9: Performance of the first instance of the models

After some tinkering, the dataset was normalized and balanced, which produced results that were closerto what was shown in the paper. More metrics were also added for a better understanding of the performance of the models, as seen in Table 11.

Table 11: Model evaluation after normalization and balancing of the dataset

Model	**Accuracy**	**Precision**	**Recall**	**F1**
SVM	96.64	82.57	73.45	75.82
KNN	75.53	80.37	65.42	67.01
GNB	96.75	82.36	72.88	75.52
Random Forest	98.11	86.06	81.17	83.54

Figure 10 showcases the confusion matrix of a Hybrid model, that took the predictions of SVM, KNN, and GNB and used them as features in the input of the Random Forest model. Due to the imbalance of the labels of the dataset, the diagonal is not significantly marked apart from the first two labels (Normal and UDP-Flood), as those are the most prominent classifications. However, upon further inspection, it is possible to see that the deviations from said diagonal are minimal, when compared to the scale of each label's dominance, which goes along the values shown in Table 11.

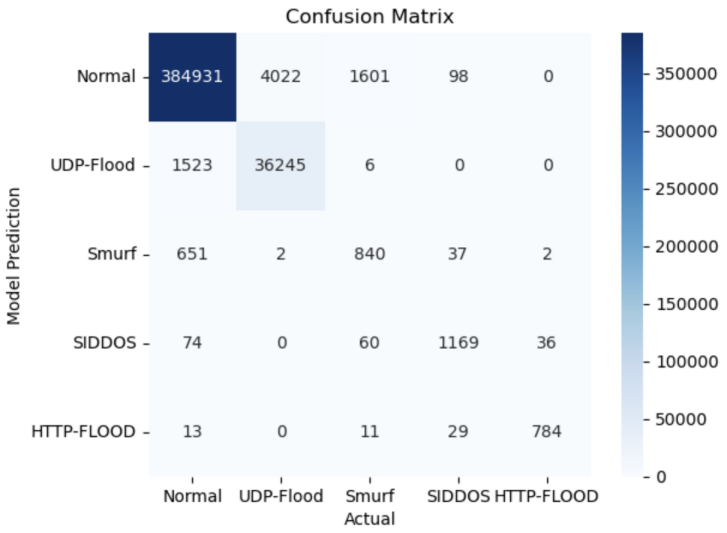


Figure 10: Random Forest confusion matrix

With the model successfully implemented and validated, changes that were outside of the work presented on the guiding material started being made.

5.3 Feature Engineering

As mentioned before, the feature engineering process was made based on feature importance: less important features are removed to increase both the speed and the accuracy of the models. To do so, 4 methods were used as described in Section 4.3.2.

5.3.1 Fisher Score

The first one was Fisher Score. This statistical method evaluates each feature comparing the mean values and variances between different classes, measureing their discriminatory power [Rosidi, 2023]. The higher the feature's score, the more separation between classes it provides, therefore, a higher feature importance is attributed to it. A screenshot of one of the outputs of Fisher Score can be seen in Figure 11.

```
PKT_RATE: 9.97%
LAST_PKT_RESEVED: 8.18%
FID: 7.54%
PKT_TYPE: 6.31%
NUMBER_OF_PKT: 6.30%
PKT_DELAY: 6.08%
BYTE_RATE: 5.93%
PKT_SIZE: 4.52%
PKT_AVG_SIZE: 4.52%
PKT_ID: 4.29%
PKT_RESEVED_TIME: 3.39%
PKT_R: 3.39%
PKT_IN: 3.39%
PKT_OUT: 3.39%
PKT_SEND_TIME: 3.39%
UTILIZATION: 3.08%
NUMBER_OF_BYTE: 3.08%
NODE_NAME_TO: 2.60%
SRC_ADD: 2.39%
DES_ADD: 2.21%
PKT_DELAY_NODE: 2.04%
NODE_NAME_FROM: 1.65%
FIRST_PKT_SENT: 1.18%
TO_NODE: 0.71%
FROM_NODE: 0.36%
SEQ_NUMBER: 0.11%
```

Figure 11: Screenshot of Fisher Score values

Using this method to begin removing features from the dataset, a detail was made clear: the feature importance for each feature was changing, but not significantly. This is due to the fact that, although the Fisher Score formula itself is independent for each feature, the statistical properties (mean and variance) used in the formula are dependent on the overall data distribution. Thus, removing one feature can change the distribution of the dataset, indirectly affecting the Fisher Scores of the remaining features, resulting in small fluctuations in the score of each feature. Table 12 shows how removing features affected the importance, as well as what features were removed (identified with a bright red cell with a slash). As for the other colors, yellow means no change occurred in the value, and green means it increased.

Table 12: Non-Hybrid models Fisher Score feature importance

Feature	27 Features	26 Features	22 Features	21 Features
SRC_ADD	2.39%	2.39%	2.49%	2.49%
DES_ADD	2.21%	2.21%	2.30%	2.30%
PKT_ID	4.29%	4.29%	4.47%	4.47%
FROM_NODE	0.36%	0.36%	/	/
TO_NODE	0.71%	0.71%	/	/
PKT_TYPE	6.31%	6.31%	6.56%	6.57%
PKT_SIZE	4.52%	4.52%	4.70%	4.71%
FLAGS	0%	/	/	/
FID	7.54%	7.54%	7.85%	7.86%
SEQ_NUMBER	0.11%	0.11%	0.11%	/
NUMBER_OF_PKT	6.30%	6.30%	6.56%	6.56%
NUMBER_OF_BYTE	3.08%	3.08%	3.20%	3.20%
NODE_NAME_FROM	1.65%	1.65%	/	/
NODE_NAME_TO	2.60%	2.60%	2.71%	2.71%
PKT_IN	3.39%	3.39%	3.53%	3.53%
PKT_OUT	3.39%	3.39%	3.53%	3.53%
PKT_R	3.39%	3.39%	3.53%	3.53%
PKT_DELAY_NODE	2.04%	2.04%	2.12%	2.13%
PKT_RATE	9.97%	9.97%	10.37%	10.38%
BYTE_RATE	5.93%	5.93%	6.17%	6.17%
PKT_AVG_SIZE	4.52%	4.52%	4.70%	4.71%
UTILIZATION	3.08%	3.08%	3.20%	3.20%
PKT_DELAY	6.08%	6.08%	6.32%	6.33%
PKT_SEND_TIME	3.39%	3.39%	3.53%	3.53%
PKT_RESEVED_TIME	3.39%	3.39%	3.53%	3.53%
FIRST_PKT_SENT	1.18%	1.18%	/	/
LAST_PKT_RESEVED	8.18%	8.18%	8.52%	8.53%

For this method, the first removal was FLAGS, as it showed a 0% feature importance, which is the result of this column being full of 0's, with no other value in it, showing zero statistical value. After that, all of the features with a value smaller than 1% were also removed, with the exception of SEQ_NUMBER. This feature was left behind because, as of this test, the other methods were being run at the same time, and this feature was deemed as important, therefore a decision was made to keep it. However, after pondering this decision, it was also removed and the methods were isolated for a more scientific analysis.

As seen in the table, with the removal of 6 features, although changes were seen in the importance of many features, the increases were minimal, with the biggest one being 0.46%. Using this method for feature evaluation, all the models either had a performance deterioration or stagnation, with KNN being an exception that had a very small boost in performance with the removal of one of the features.

5.3.2 Model Coefficients

For the coefficient analysis to be possible, the parameter 'kernel' from the SVM model, had to be changed to linear. This not only allowed for the function 'coef_' to be utilized but also improved the performance of the model, as shown in Figure 12.

Kernel: Sigmoid

```
[16:17:59] Started training SVM... Finished! Time elapsed: 1s
[16:18:00] Started predicting... Finished! Time elapsed: 45s

Accuracy: 62.99943998852208%
Precision: 73.30500834267804%
Recall: 44.958807500416555%
F1: 43.1141971330731%
```

Kernel: Linear

```
[16:13:43] Started training SVM... Finished! Time elapsed: 14s
[16:13:57] Started predicting... Finished! Time elapsed: 27s

Accuracy: 96.83315823332578%
Precision: 82.44518671128O7%
Recall: 69.11859849162066%
F1: 72.40380366249957%
```

Figure 12: Performance comparison between the different kernel options

In a multi-class SVM, Scikit-learn uses a one-vs-one strategy by default, creating a binary classifier for every pair of classes. This means that the resulting table will have $\frac{K\times(K-1)}{2}$, where K is the number of classes. Given that the dataset has 5 classes, the coefficient table for this model will have $\frac{5\times4}{2} = 10$ rows, where each row corresponds to a unique pair of classes. This is not specified in the documentation of the library online, however, upon further inspection of the source code for the library, the function shown in Figure 13 was found to be the deciding factor for the order of the class distribution.

This means that when requesting the coefficients for a linear kernel SVM model, the order of the rows is obtained from a sequential distribution of classes. In this case, where the dataset had the classes [Normal, UDP-Flood, Smurf, SIDDOS, HTTP-FLOOD], the coefficient table is shown in Table 13.

```
def _one_vs_one_coef(dual_coef, n_support, support_vectors):
    """Generate primal coefficients from dual coefficients
    for the one-vs-one multi class LibSVM in the case
    of a linear kernel."""

    # get 1vs1 weights for all n*(n-1) classifiers.
    # this is somewhat messy.
    # shape of dual_coef_ is nSV * (n_classes -1)
    # see docs for details
    n_class = dual_coef.shape[0] + 1

    # XXX we could do preallocation of coef but
    # would have to take care in the sparse case
    coef = []
    sv_locs = np.cumsum(np.hstack([[0], n_support]))
    for class1 in range(n_class):
        # SVs for class1:
        sv1 = support_vectors[sv_locs[class1] : sv_locs[class1 + 1], :]
        for class2 in range(class1 + 1, n_class):
            # SVs for class1:
            sv2 = support_vectors[sv_locs[class2] : sv_locs[class2 + 1], :]

            # dual coef for class1 SVs:
            alpha1 = dual_coef[class2 - 1, sv_locs[class1] : sv_locs[class1 + 1]]
            # dual coef for class2 SVs:
            alpha2 = dual_coef[class1, sv_locs[class2] : sv_locs[class2 + 1]]
            # build weight for class1 vs class2

            coef.append(safe_sparse_dot(alpha1, sv1) + safe_sparse_dot(alpha2, sv2))
    return coef
```

Figure 13: Source code of the '_one_vs_one_coef' function

Table 13: Partial table with the coefficients of the SVM model

	SRC_ADD	DES_ADD	PKT_ID	PKT_TYPE	PKT_SIZE	FID	NUMBER_OF_PKT	NUMBER_OF_BYTE	NODE_NAME_TO	PKT_IN	PKT_OUT
Normal vs UDP-Flood	-0.007440	-0.006614	0.001290	-0.312250	0.012600	0.014083	-0.084206	-0.033877	0.000055	-0.017516	-0.017468
Normal vs Smurf	0.002291	0.001713	0.001306	0.396829	-0.124828	-0.004234	0.281627	0.104168	-0.000006	-0.006371	-0.006337
Normal vs SIDDOS	0.179476	0.146672	0.000064	0.969935	0.103511	-0.307535	0.175556	0.200820	0.000013	-0.001095	-0.001095
Normal vs HTTP-FLOOD	0.440983	-0.676545	0.452996	0.204324	-0.160083	0.397797	0.072766	0.346774	0.000176	-0.011965	-0.011965
UDP-Flood vs Smurf	0.006783	0.005924	-0.001248	0.310661	-0.123523	-0.012871	0.083662	0.038042	-0.000005	0.011373	0.011323
UDP-Flood vs SIDDOS	0.012354	0.010507	0.000875	0.632549	0.005596	-0.022728	0.451948	0.114108	-0.000019	0.004002	0.003964
UDP-Flood vs HTTP-FLOOD	0.171380	-0.372022	0.033496	0.511949	-0.091777	0.058535	0.086522	0.232952	0.000060	-0.001363	-0.001363
Smurf vs SIDDOS	0.177924	0.145458	-0.000051	0.959715	0.150570	-0.304956	0.147490	0.231800	-0.000024	0.002294	0.002295
Smurf vs HTTP-FLOOD	0.012339	-1.000000	0.711333	0.423597	-0.169114	0.041263	-0.118025	0.357732	-0.000025	-0.016608	-0.016609
SIDDOS vs HTTP-FLOOD	0.082327	0.069579	-0.025732	0.075112	-0.018886	0.078242	0.316969	0.028385	0.000013	-0.007125	-0.007125

In the table, some cells are highlighted in red and green. These are the ones in which their absolute value is bigger (green if positive, red if negative) than the average of the absolute values of that column. Since a bigger value indicates a bigger weight on the decision of the model (if it belongs to a class or not), the more marked (therefore relevant) cells a column has, the bigger their feature importance should be.

The same was done for GNB, using 'theta_', the equivalent function to get the coefficients. In this case, however, each row corresponds to one class, making it more straightforward:

Table 14: Partial table with the coefficients of the GNB model

	SRC_ADD	DES_ADD	PKT_ID	PKT_TYPE	PKT_SIZE	FID	NUMBER_OF_PKT	NUMBER_OF_BYTE	NODE_NAME_TO	PKT_IN	PKT_OUT
Normal	0.070205	-0.382080	0.556287	-0.030010	-0.274462	0.115121	0.549473	0.199871	-0.370383	-0.245630	-0.245747
UDP-Flood	-0.266547	0.151817	-0.246367	0.575226	-0.261932	0.806884	0.138330	0.245850	0.022681	-0.072785	-0.072847
Smurf	-0.008887	-0.196220	0.231922	0.348892	0.281396	-0.086795	0.098324	0.198221	-0.175690	-0.175287	-0.175270
SIDDOS	0.439450	0.225993	-0.198360	-0.435628	-0.294552	-0.402341	-0.066605	-0.508154	0.414122	0.593701	0.593607
HTTP-FLOOD	-0.234221	0.200490	-0.343482	-0.458480	0.549550	-0.432868	-0.719522	-0.135788	0.109270	-0.099999	-0.099743

Looking at the table for this model, Table 14, it isn't as clear which ones are and are not contributing much to the predictions, as there are no columns that don't have any color. In this case, it would be necessary to discard the ones that contributed the least.

5.3.3 Random Forest Feature Importance

The final method was the feature importance provided by the Random Forest model directly, as shown in Figure 14. As mentioned before, the same process of removing features used for Fisher Score was used here, with 'feature_importances_' giving the values this time.

For the hybrid model, the size was reduced from 26+3(with the +3 being the other model's inputs) to 15+2. This also resulted in the removal of SVM predictions, which took a long time to train and predict for a performance worse than the other models and also limited the Hybrid one, as removing it improved the scores. Table 15 shows the progression of importance when removing features. This time, a new color has been added, light red, because this approach takes into consideration the model, making it possible for an importance to go down.

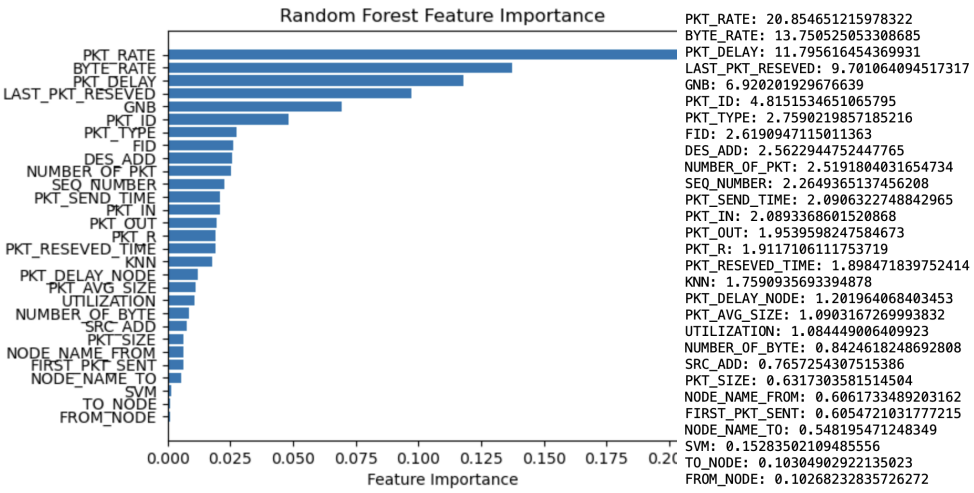


Figure 14: Screenshot of Random Forest feature importance

Table 15: Hybrid model feature importance

Feature	27+3 Features	26+3 Features	22+2 Features	19+2 Features	15+2 Features
SRC_ADD	0.77%	0.77%	0.76%	/	/
DES_ADD	2.56%	2.56%	2.16%	3.02%	4.19%
PKT_ID	4.82%	4.82%	3.57%	4.70%	3.78%
FROM_NODE	0.10%	0.10%	/	/	/
TO_NODE	0.10%	0.10%	/	/	/
PKT_TYPE	2.76%	2.76%	2.05%	1.81%	2.03%
PKT_SIZE	0.63%	0.63%	1.24%	0.87%	/
FLAGS	0.00%	/	/	/	/
FID	2.62%	2.62%	1.97%	2.92%	2.34%
SEQ_NUMBER	2.26%	2.26%	1.99%	2.23%	2.60%
NUMBER_OF_PKT	2.52%	2.52%	3.34%	2.27%	3.77%
NUMBER_OF_BYTE	0.84%	0.84%	1.36%	0.80%	/
NODE_NAME_FROM	0.61%	0.61%	/	/	/
NODE_NAME_TO	0.55%	0.55%	0.41%	/	/
PKT_IN	2.09%	2.09%	1.96%	2.25%	2.31%
PKT_OUT	1.95%	1.95%	1.96%	2.12%	2.18%
PKT_R	1.91%	1.91%	2.00%	2.12%	2.22%
PKT_DELAY_NODE	1.20%	1.20%	0.52%	/	/
PKT_RATE	20.85%	20.85%	22.67%	18.95%	24.19%
BYTE_RATE	13.75%	13.75%	14.40%	16.27%	15.87%
PKT_AVG_SIZE	1.09%%	1.09%%	1.02%	0.94%	/
UTILIZATION	1.08%	1.08%	2.08%	0.79%	/
PKT_DELAY	11.80%	11.80%	8.92%	11.02%	8.05%
PKT_SEND_TIME	2.09%	2.09%	1.93%	1.82%	1.82%
PKT_RESEVED_TIME	1.90%	1.90%	1.87%	1.88%	2.00%
FIRST_PKT_SENT	0.61%	0.61%	/	/	/
LAST_PKT_RESEVED	9.70%	9.70%	12.06%	12.92%	13.23%
SVM	0.15%	0.15%	/	/	/
KNN	1.76%	1.76%	1.36%	1.23%	1.30%
GNB	6.92%	6.92%	8.39%	9.07%	8.14%

5.4 Discussion

All of these modifications in the dataset had implications for the models' performance. Not every change was beneficial to all models, except for the hybrid one. Table 16 shows how each of the previous sets of features affected the models. For an easier read of the table, the cells have been painted, following the previous color logic of the previous table.

Table 16: Feature results comparison

Number of Features		SVM				KNN				GNB				Random Forest			
Non-Hybrid	Hybrid	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
27	27+3	96.64	82.57	73.45	75.82	75.53	80.37	65.42	67.01	96.75	82.36	72.88	75.52	98.11	86.06	81.17	83.54
26	26+3	96.64	82.57	73.45	75.82	75.53	80.37	65.42	67.01	96.75	82.36	72.88	75.52	98.11	86.06	81.17	83.54
22	22+2	96.86	82.57	73.50	75.98	75.21	80.29	66.79	68.14	96.79	82.34	73.19	75.77	98.28	88.56	81.68	84.98
21	19+2	93.83	82.45	69.12	72.40	75.23	80.35	67.03	68.31	96.79	82.25	67.58	71.35	98.37	90.53	81.83	85.96
21	15+2	93.83	82.45	69.12	72.40	75.23	80.35	67.03	68.31	96.79	82.25	67.58	71.35	98.40	90.85	81.79	86.08

The results shown in the first implementation of the models were promising but, as mentioned, were not realistic for two reasons: the dataset was not balanced, and the only metric that was being used to measure its performance was accuracy, which is not sufficient to tell how the model is performing. After balancing and normalizing the data as well as adding some more evaluation metrics, the results, shown in Table 11, were more informative.

The feature engineering provided a huge improvement in the performance of the models. The information retrieved from the coefficients still needs to be implemented into the process, however, using Fisher Score and the Random Forest Feature Importance, the performance of the hybrid model improved, and the amount of features was also reduced.

Although these results are still using a dataset and not real information extracted from a simulated cloud service, the preliminary conclusion taken from the metrics is that the performance of the models in that scenario will be, at least, relevant enough to be usable.

The methodology used for this first method showed a lot of limitations, some coming from a lack of experience, others imposed by time restrictions. The following identified limitations for this method were taken into account when developing the second one.

- **Missing cross-validation**: This solution had no cross-validation implementation in the evaluation of the models. This might have lead to biased results which is not desirable when training models.
- **Feature Engineering**: Not all methods of feature engineering that were proposed were implemented. These were also not thoroughly nor scientifically tested, leading to possible inaccurate conclusions.
- **Anomalies**: The dataset used to train the models only showcased DDoS anomalies, which is extremely limiting to the model applications.

- **Result Analysis**: The whole methodology used for this first method was very experimental, with little to no thought given to result analysis. A lot of the experiments were run with many variables changing between them, resulting in not very methodical processes.

5.5 Summary

In this chapter, the results obtained for the first method were shown, such as the selected models and the dataset that was used to train them. The implementation of those models as well as how they were validated was also addressed, as well as the process of feature engineering that was applied to improve the results. Finally, the discussion of the results touched upon the conclusions that were taken from these results, mentioning their limitations.

Chapter 6

Cross-Validated CoRF Results and Discussion

In this chapter, the final results of the thesis will be presented and discussed. First, the new dataset introduced will be addressed, followed by the results obtained from this new method across all datasets. The practical implications of said results will also be addressed, as well as the limitations of the methodology.

6.1 New Datasets

For this method, two new datasets were introduced. The first one is the EdgeIIoT dataset, which is an extensive collection of data sourced from various IoT devices and sensors deployed within an edge computing framework. This dataset consists of a wide range of data types, including sensor readings, device status logs, and network performance metrics, making it a versatile resource for researchers and developers working on edge computing and IoT applications [Ferrag, 2022]. Although these models are not planned to be trained with network protocol features (which is the case in this dataset), this dataset aims to test their performance with these types of features, as there is also a possibility that they can be extracted. This also helps test how the models would behave in edge computing scenarios, in case they needed to be introduced in such conditions.

This dataset comes divided in two, one that is meant for Classical Machine Learning, consisting of a smaller sample, and one meant for Deep Neural Networks, that has a bigger amount of data, as these usually require more information, which have both been used to test the model.

The other dataset that was introduced was requested in collaboration between Instituto Pedro Nunes and the University of Coimbra, as the feature extraction mechanism was not ready at the time of the delivery of this thesis. This dataset was extracted in a very similar way to what was planned for the Nexus project and, having many tools in common.

The purpose of this dataset was to test the performance of these models in aspects

that were missing on the other two datasets, such as CPU usage, memory usage, container information, etc. It also functions as a replacement of the dataset that was meant to be created using the feature extraction mechanism that was being developed parallel to this thesis, as it was created in very similar conditions, as mentioned previously.

6.2 Results

In this section, the final results of the tests will be presented. For each dataset, the data distribution will be given for each of the balancing methods, and, for each model, the table of results will be shown, along with the time taken to fit and score each one. Graphs comparing the diverse approaches will also be shown.

6.2.1 DDoS Dataset

This is the same dataset that was used in the first method. On the tables, the Features column indicates which balancing method and which feature removal process was used. For example, Original_RF<1% indicates that the dataset was not balanced (keeping the original distribution) and the feature removal technique used was the Random Forest Feature Importance where the removed ones were the ones under 1%. As mentioned before, the dataset was slightly altered to include a column that binarily identified the attacks. Figure 32 shows the original data distribution of the dataset.

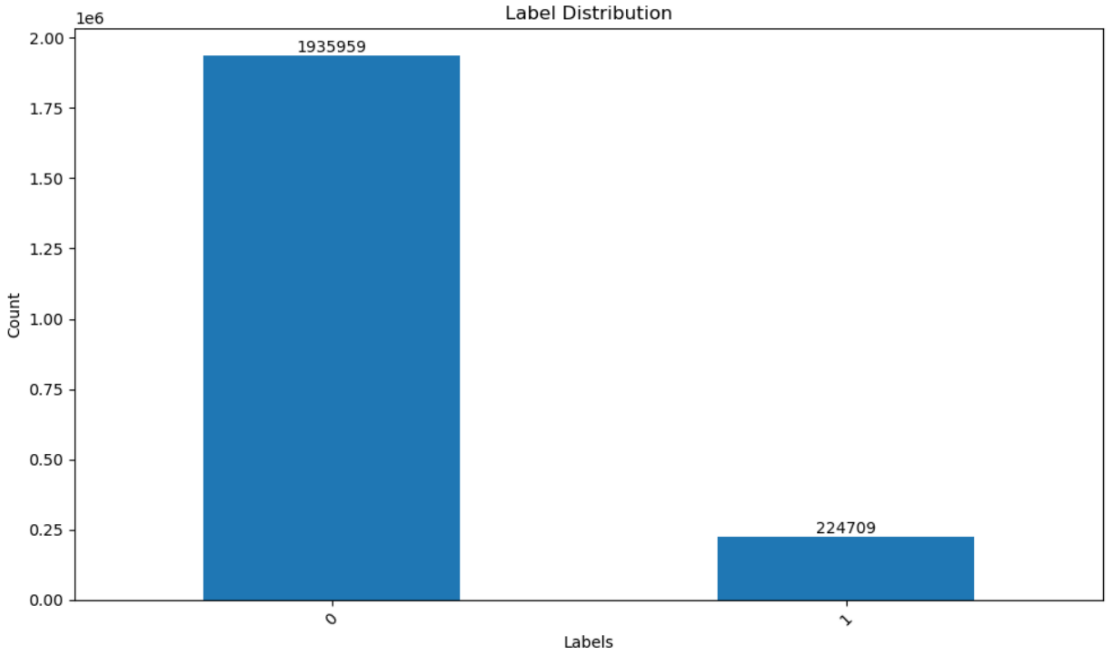


Figure 15: Original data distribution of the DDoS Dataset

To balance the dataset, the limited approach will reduce the number of normal

traffic to 224709, and the Smote approach will increase the number of malicious traffic to 1935959.

6.2.1.1 KNN

Table 17: Table of the KNN results on the DDoS Dataset

KNN						
Features	Accuracy	Precision	Recall	F1	Fitting(s)	Scoring(s)
Original_Original	98,66%	99,19%	93,63%	96,20%	0,27	420,71
Original_RF<1%	98,66%	99,19%	93,64%	96,29%	0,23	397,38
Original_Correlations	98,66%	99,19%	93,64%	96,29%	4,91	20,05
Original_Coefficients	98,66%	99,19%	93,64%	96,29%	3,86	16,19
Limited_Original	93,09%	93,65%	93,09%	93,07%	0,05	21,6
Limited_RF<1%	93,10%	93,66%	93,10%	93,08%	0,04	20,01
Limited_Correlations	93,10%	93,66%	93,10%	93,08%	0,72	4,63
Limited_Coefficients	93,12%	93,68%	93,12%	93,09%	0,48	4,31
Balanced_Original	95,32%	95,49%	95,32%	95,31%	0,65	1247,91
Balanced_RF<1%	95,09%	95,31%	95,09%	95,09%	0,58	1243,5
Balanced_Correlations	94,12%	94,55%	94,12%	94,11%	9,45	35,84
Balanced_Coefficients	94,96%	95,20%	94,96%	94,96%	7	28,35

Right away, in Table 17, it is possible to see the difference between the binary attack identification and the normal one. This can be seen in the first row: with the first method, the KNN model obtained Accuracy, Precision, Recall, and F1 scores of 75.53, 80.37, 65.42, and 67.01, respectively, while using the binary identification, the scores were 98.66, 99.19, 93.63, and 96.20, respectively.

Figure 16 shows the performance graph of the KNN model on each set of conditions. it shows that the performance is about the same, however, if the main goal of the model implementation is to squeeze the maximum amount possible from the scores, this scale is not good for that. In Figure 17, the scale of the graph has been changed from 0 to 100 to 80 to 100.

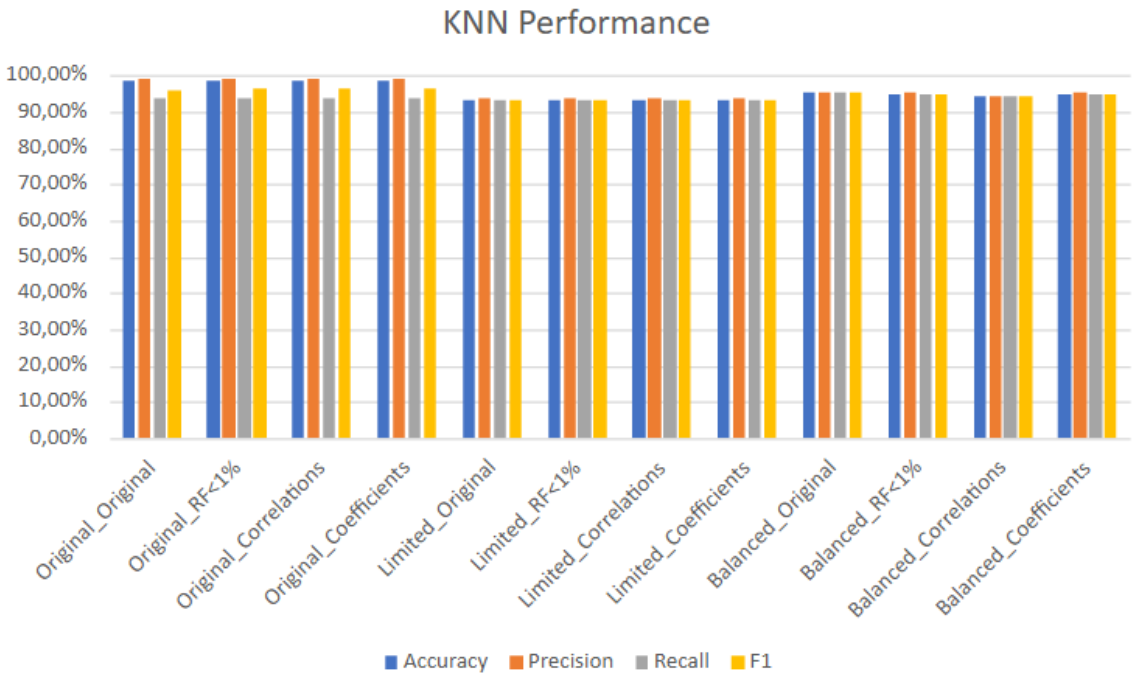


Figure 16: Graph of the KNN results on the DDoS Dataset

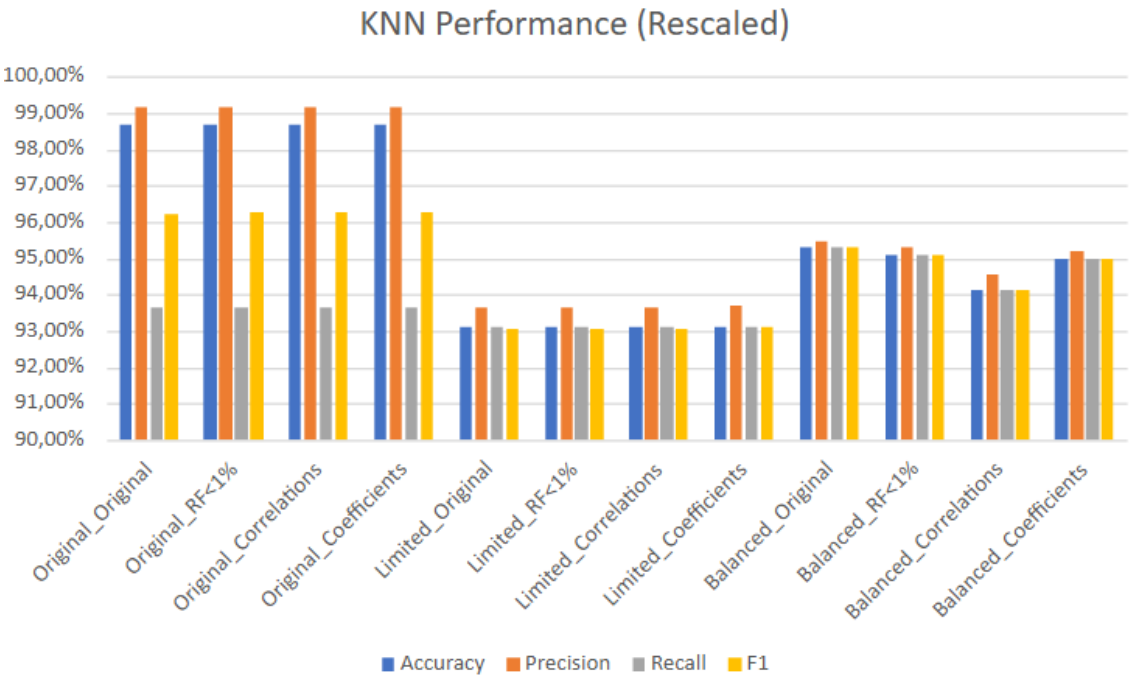


Figure 17: Rescaled graph of the KNN results on the DDoS Dataset

Here we can see that the best performance of the model is with its original distribution, with the only advantage of balancing the data being the Recall when balancing using Smote. Other than that, the only difference is that the scores get a lot closer together, becoming more consistent with each other. If the main goal is speedy identification, however, limiting the data might be a good choice, as seen in Figure 18.

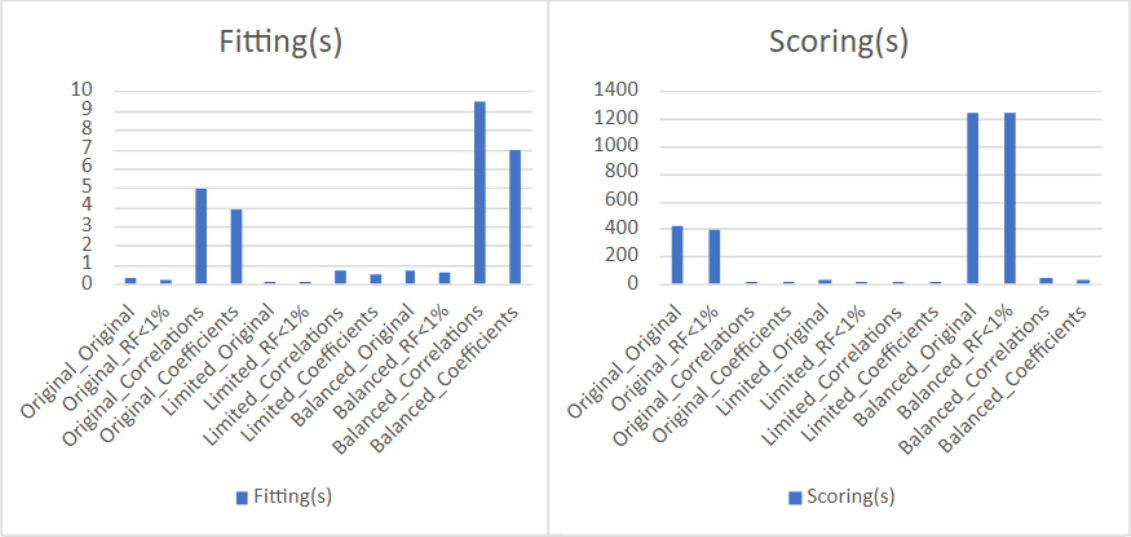


Figure 18: Fitting and Scoring graph of the KNN results on the DDoS Dataset

Here we see that, with a hit of about 5 to 6% on the accuracy and precision of the model, the scoring time can go from around 400 seconds to around 20 when using the original feature set, but this time can go even lower when using the feature removal techniques. Another advantage of these feature removal techniques that can be seen is that, when using the original features or the ones indicated by the Random Forest Feature Importance, the model takes less time to fit and more time to score, while when using the Coefficients or Correlations approach, it takes more time to fit but the scoring becomes much faster, all of this while keeping the same performance. This can also be relevant for many implementations of the model, as usually, a faster prediction is more important that a fast training.

6.2.1.2 GNB

In Table 18, it is once again possible to see the improvement of the binary identification, going from 96.75, 82.32, 72.88, and 75.52 to 97.41, 98.58, 87.56, and 92.18, for accuracy, precision, recall and f1 score, respectively.

Table 18: Table of the GNB results on the DDoS Dataset

GNB						
Features	Accuracy	Precision	Recall	F1	Fitting(s)	Scoring(s)
Original_Original	97,41%	98,58%	87,56%	92,18%	0,57	0,33
Original_RF<1%	97,41%	98,58%	87,56%	92,18%	0,49	0,31
Original_Correlations	89,96%	94,61%	51,72%	50,67%	0,34	0,23
Original_Coefficients	97,40%	98,58%	87,53%	92,16%	0,29	0,22
Limited_Original	91,63%	92,83%	91,63%	91,57%	0,11	0,06
Limited_RF<1%	91,63%	92,83%	91,63%	91,57%	0,1	0,06
Limited_Correlations	92,08%	93,16%	92,08%	92,03%	0,07	0,04
Limited_Coefficients	91,63%	92,83%	92,63%	91,57%	0,06	0,04
Balanced_Original	91,64%	92,84%	91,64%	91,58%	1,13	0,65
Balanced_RF<1%	91,62%	92,82%	91,62%	91,56%	0,98	0,59
Balanced_Correlations	92,05%	93,14%	92,05%	91,99%	0,69	0,48
Balanced_Coefficients	91,64%	92,84%	91,64%	91,58%	0,57	0,43

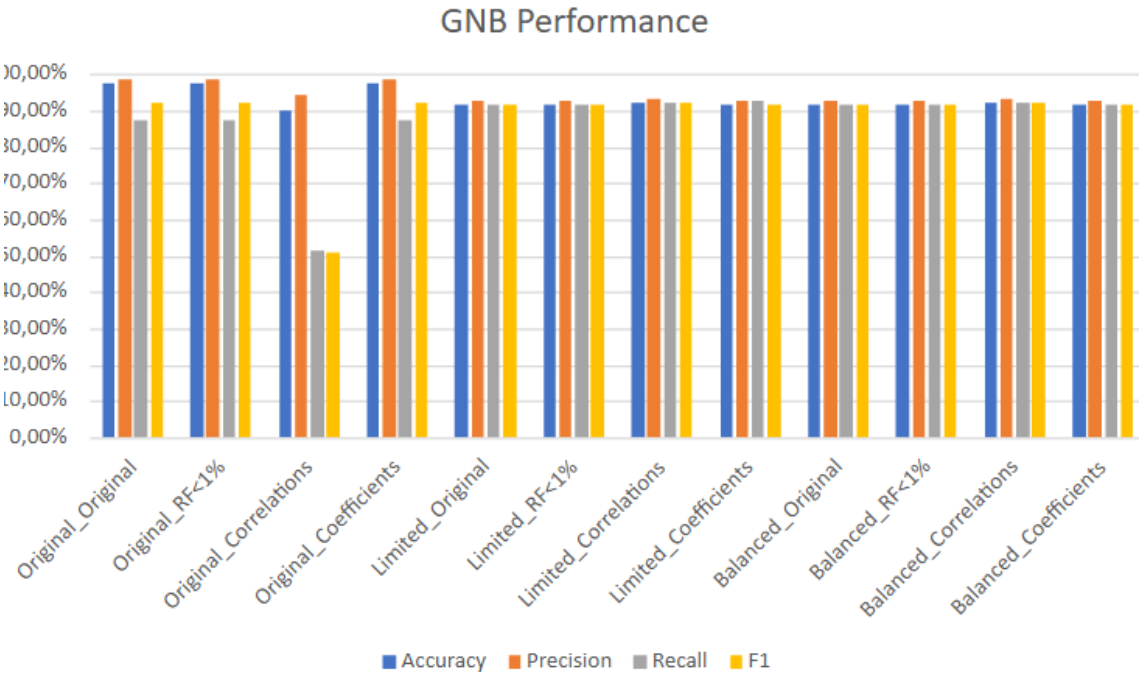


Figure 19: Graph of the GNB results on the DDoS Dataset

Figure 19 keeps the previous trend, showing more stable results in the balanced versions, but only improving the Recall score. The best overall performance is still from the original distribution. We can also see that the performance gain between the limited and the balanced version is non-existent, meaning that this model probably does not benefit from more information.

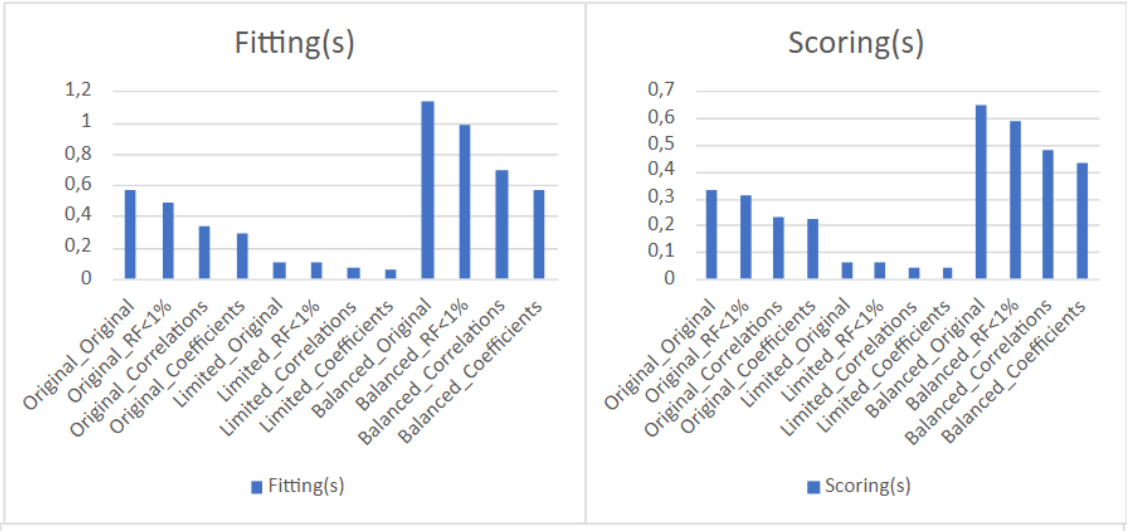


Figure 20: Fitting and Scoring graph of the GNB results on the DDoS Dataset

The times seen in Figure 20 show the potential of the feature removal techniques. The time required to fit and score drastically decreases while the performance of the models stays the same, with the exception of the Correlations method in the original distribution.

6.2.1.3 Random Forest

Table 19: Table of the Random Forest results on the DDoS Dataset

RandomForest						
Features	Accuracy	Precision	Recall	F1	Fitting(s)	Scoring(s)
Original_Original	98,28%	97,18%	93,43%	95,21%	159,29	3,54
Original_RF<1%	98,60%	98,83%	93,60%	96,03%	198,78	3,98
Original_Correlations	97,56%	93,75%	93,06%	93,40%	72,19	3,34
Original_Coefficients	98,39%	97,70%	93,49%	95,47%	99,19	3,21
Limited_Original	91,03%	91,18%	91,03%	91,03%	17,81	0,48
Limited_RF<1%	91,46%	91,67%	91,46%	91,45%	20,87	0,48
Limited_Correlations	89,46%	89,47%	89,46%	89,46%	7,45	0,48
Limited_Coefficients	89,96%	90,00%	89,96%	89,95%	11,18	0,45
Balanced_Original	98,52%	98,52%	98,52%	98,52%	170,65	4,57
Balanced_RF<1%	98,55%	98,57%	98,55%	98,55%	193,96	4,76
Balanced_Correlations	94,62%	94,65%	94,62%	94,62%	97,25	6,41
Balanced_Coefficients	98,30%	98,32%	98,30%	98,30%	122,14	4,43

The improvement of the binary identification can't be fully seen here, as the previous method only used the Random Forest model in a hybrid solution, however,

even without the additional information provided by the other models in that solution, this standalone version of the Random Forest model is better, improving the Accuracy, Precision, Recall, and F1 Scores from 98.11, 86.06, 81.71, and 83.54 to 98.28, 97.18, 93.43, and 95.21, respectively.

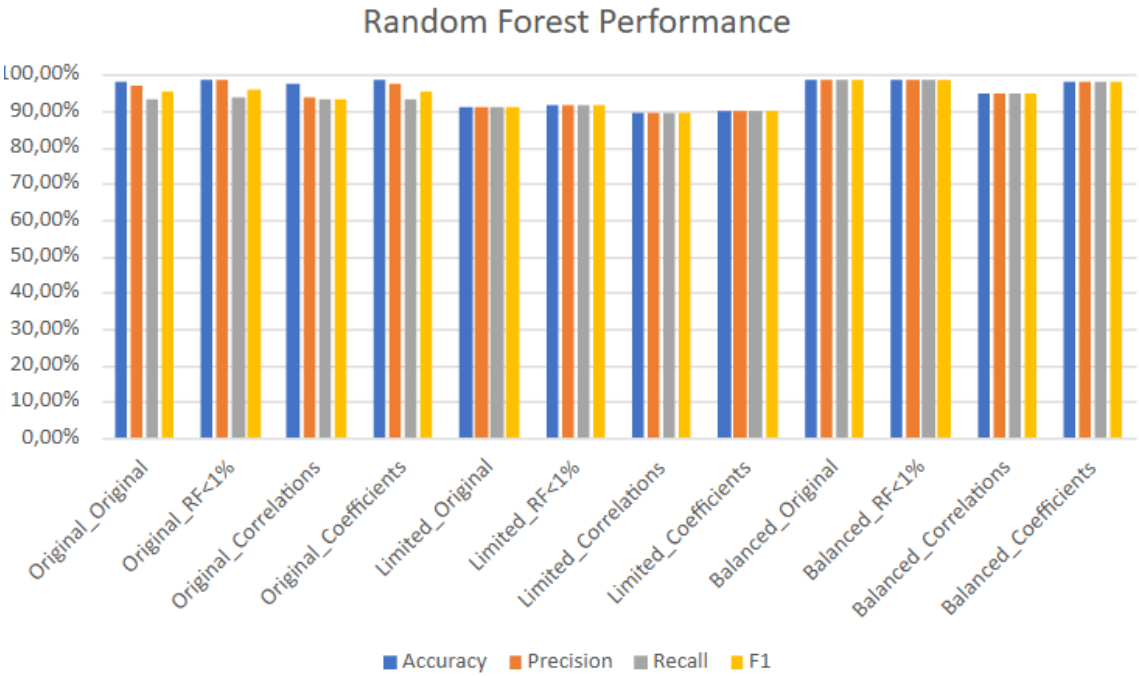


Figure 21: Graph of the Random Forest results on the DDoS Dataset

Random Forest is the first model that showed a complete improvement from a balanced dataset, specifically from the one balanced with Smote, as seen in Figure 21. Figure 22 shows a rescaled version of the graph for a better visual of the performance loss and gain for each set of conditions.

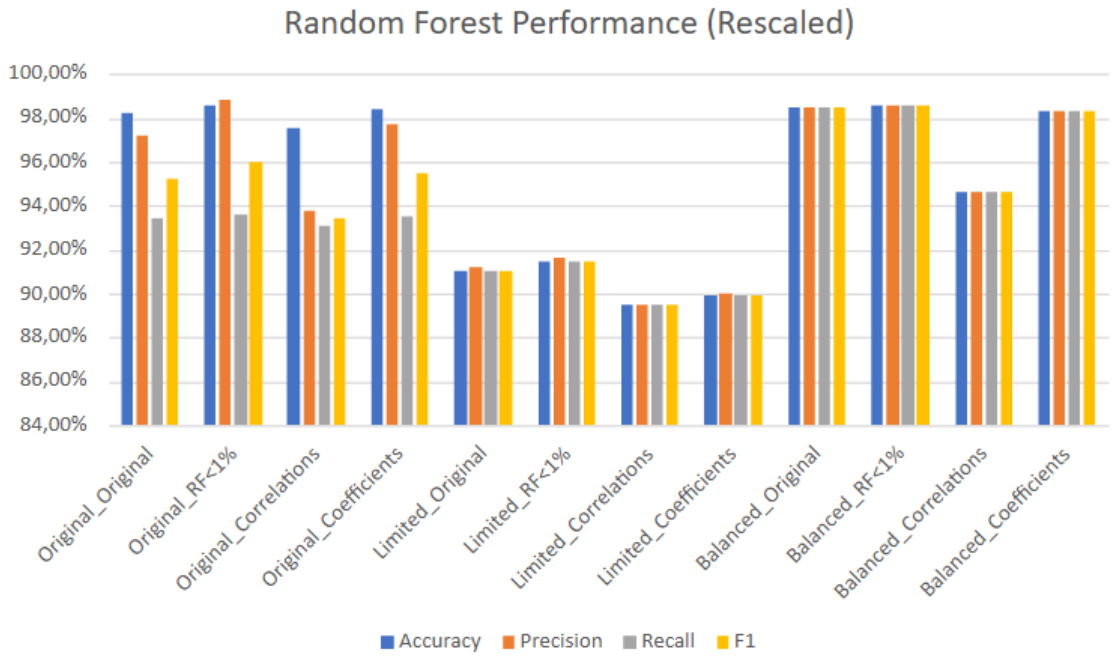


Figure 22: Rescaled graph of the Random Forest results on the DDoS Dataset

Once again the Correlations technique shows a slightly bigger loss of performance when compared with the other techniques. This can be due to a low threshold, resulting in the loss of relevant columns that have a big correlation between themselves, but not too big that makes them irrelevant.

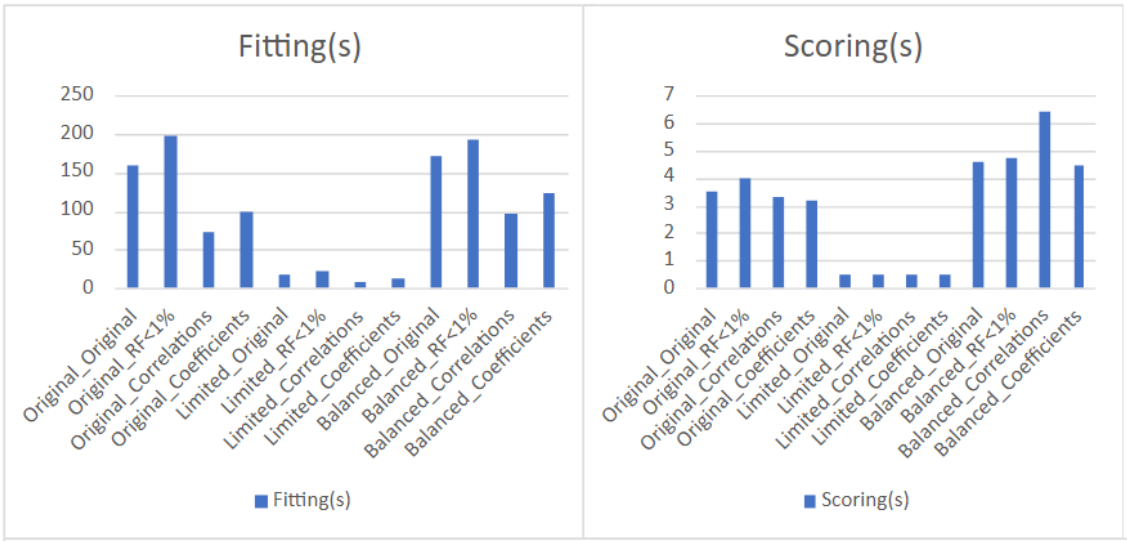


Figure 23: Fitting and Scoring graph of the Random Forest results on the DDoS Dataset

In Figure 23 the time gain of limiting the dataset is once again seen, but what is also noticed is that removing the features using the Random Forest Feature Importance approach actually increases both the time to fit and score the data.

6.2.2 EdgeIIoT-ML Dataset

The second dataset was the smallest one out of the two that come in the EdgeIIoT folder. This one is meant for classical machine learning models, as they typically use less data than neural networks. Figure 24 shows the original data distribution of the dataset.

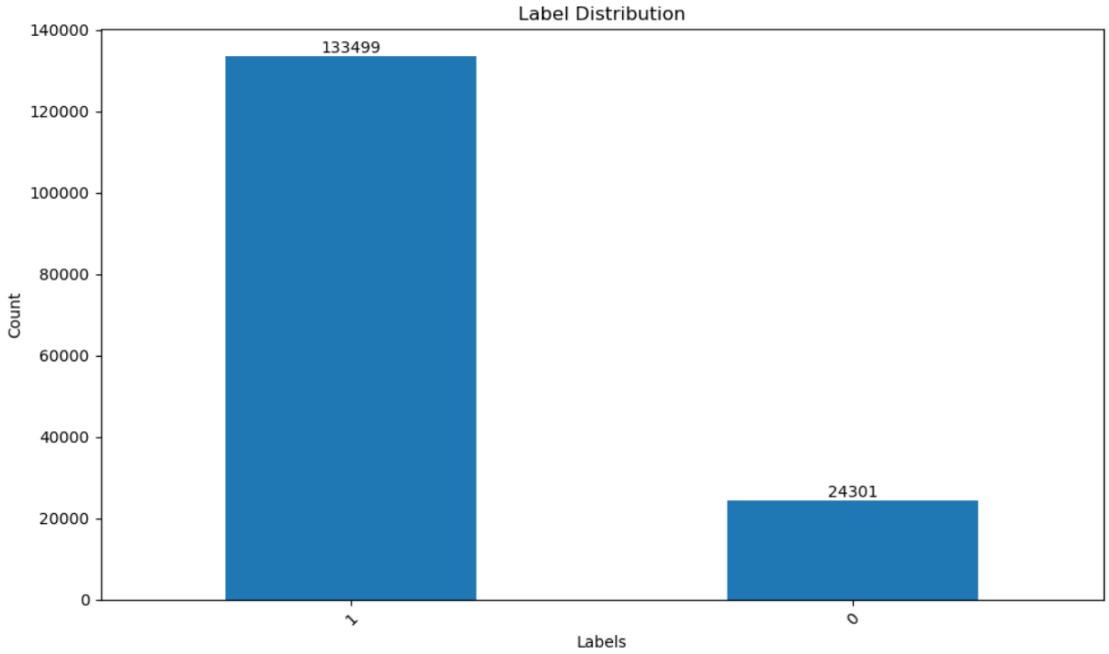


Figure 24: Original data distribution of the EdgeIIoT-ML Dataset

This time, the limited approach will reduce the number of normal traffic to 24301 instances, while the Smote approach will increase the number of malicious traffic to 133499 instances.

6.2.2.1 KNN

Table 20: Table of the KNN results on the EdgeIIoT-ML Dataset

KNN						
Features	Accuracy	Precision	Recall	F1	Fitting(s)	Scoring(s)
Original_Original	87,91%	79,14%	68,27%	71,76%	0,03	5,01
Original_Correlations	87,84%	78,94%	68,12%	71,58%	0,02	4,54
Original_RF<1%	87,89%	79,11%	68,22%	71,71%	0,17	2,87
Original_Coefficients	87,89%	79,11%	68,22%	71,71%	0,1	2,21
Limited_Original	73,46%	73,70%	73,46%	73,39%	0,01	1,01
Limited_Correlations	73,66%	73,85%	73,66%	73,61%	0,14	0,96
Limited_RF<1%	63,10%	64,54%	63,10%	62,15%	0,05	0,98
Limited_Coefficients	62,29%	63,61%	62,29%	61,35%	0,03	0,98
Balanced_Original	84,70%	84,76%	84,70%	84,69%	0,07	10,29
Balanced_Correlations	84,60%	84,66%	84,60%	84,60%	0,06	9,44
Balanced_RF<1%	84,55%	84,61%	84,55%	84,55%	0,35	4,24
Balanced_Coefficients	84,57%	84,63%	84,57%	84,56%	0,18	4,07

Contrary to the previous dataset, this one shows that balancing the data using Smote improves the performance of the models even on the KNN model, as seen in Figure 20. Although the accuracy takes a slight hit, all of the other metrics go up. This is easier to spot in Figure 25.

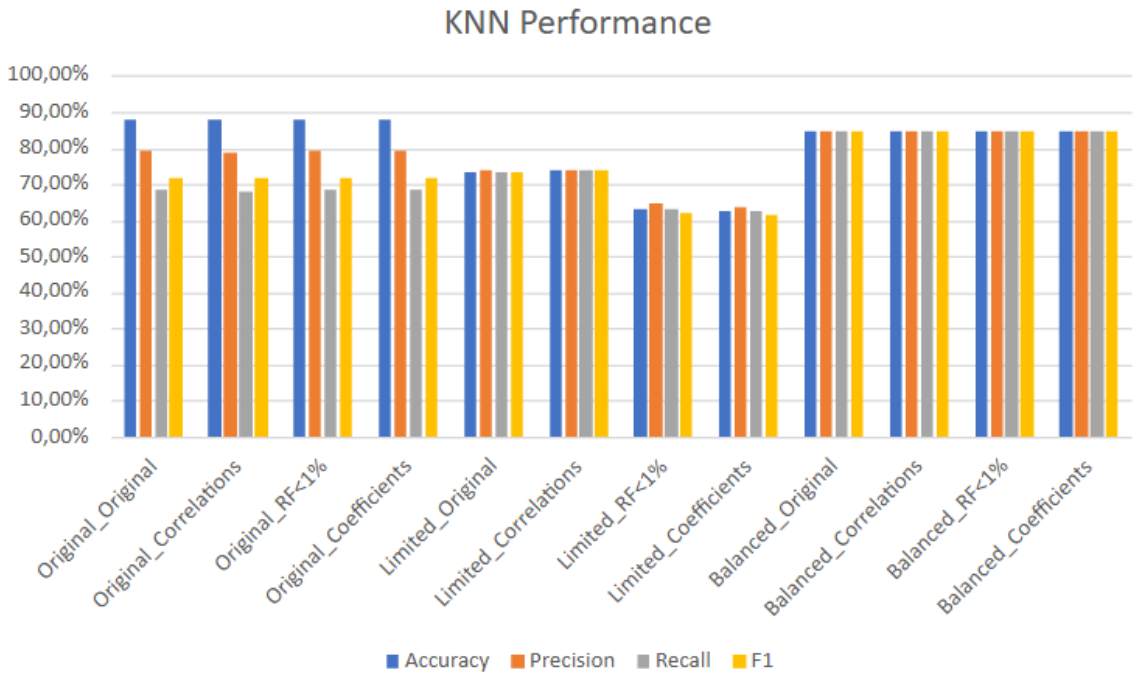


Figure 25: Graph of the KNN results on the EdgeIIoT-ML Dataset

On the other hand, the trend seen in the previous dataset continues, as the metrics get a lot more consistent when the dataset is balanced.

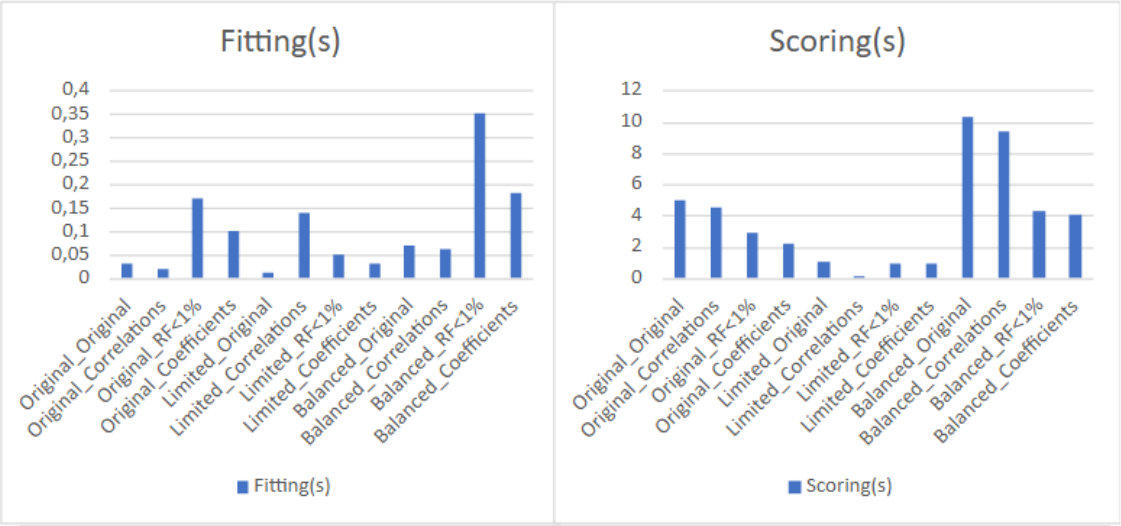


Figure 26: Fitting and Scoring graph of the KNN results on the EdgeIIoT-ML Dataset

In Figure 26 it is again seen the trade-off of increasing fitting time to reduce the scoring time when using either the Correlations or the Coefficients approach.

6.2.2.2 GNB

Table 21: Table of the GNB results on the EdgeIIoT-ML Dataset

GNB						
Features	Accuracy	Precision	Recall	F1	Fitting(s)	Scoring(s)
Original_Original	30,27%	57,40%	57,56%	30,27%	0,07	0,04
Original_Correlations	30,04%	57,76%	57,76%	30,04%	0,05	0,03
Original_RF<1%	83,85%	63,57%	54,29%	54,56%	0,02	0,02
Original_Coefficients	67,44%	54,21%	56,77%	54,54%	0,02	0,01
Limited_Original	57,70%	70,88%	57,70%	49,77%	0,02	0,02
Limited_Correlations	57,91%	72,19%	57,92%	49,85%	0,02	0,01
Limited_RF<1%	53,97%	64,11%	53,97%	43,89%	0,01	0,01
Limited_Coefficients	52,03%	59,24%	52,04%	40,41%	0,01	0,01
Balanced_Original	57,67%	70,80%	57,67%	49,74%	0,14	0,07
Balanced_Correlations	57,79%	72,00%	57,79%	49,66%	0,11	0,05
Balanced_RF<1%	54,06%	64,36%	54,06%	44,01%	0,05	0,03
Balanced_Coefficients	51,95%	58,95%	51,95%	40,26%	0,03	0,02

Table 21 shows that the performance of the GNB model is not only way below the KNN model but also way below its performance on the previous dataset. This can be because of various reasons: the reduced number of lines, and therefore information, could be impacting the model, or the different nature of the two datasets, as this data consists of protocol information while the other one uses packet information could make this model less useful for this type of data.

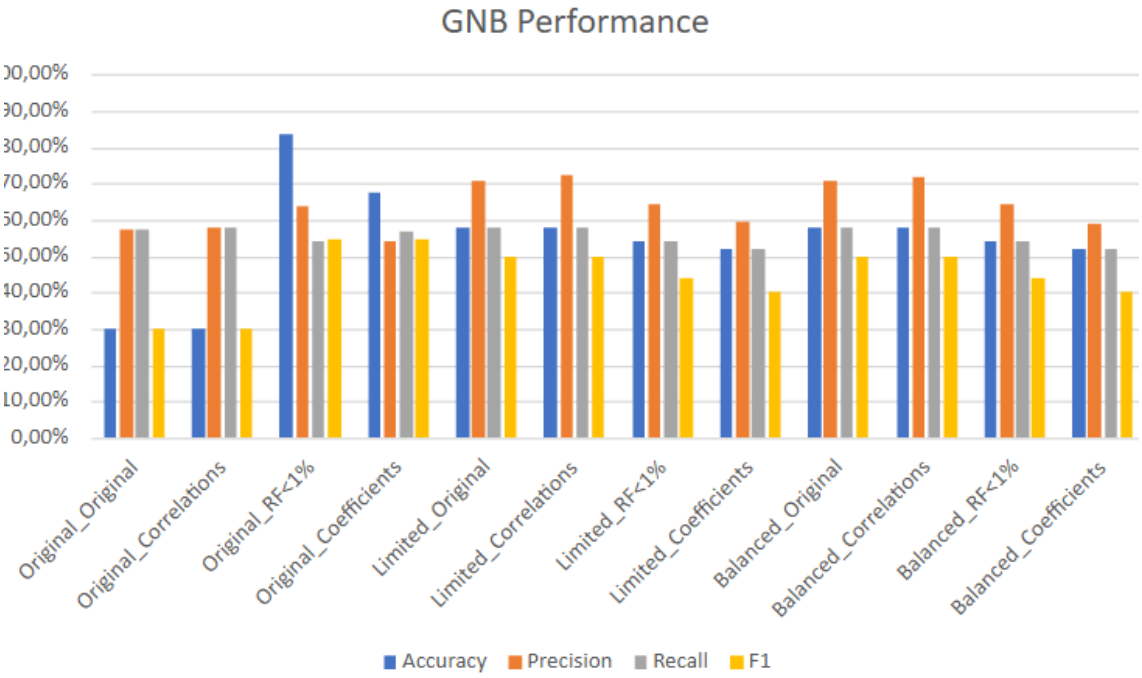


Figure 27: Graph of the GNB results on the EdgeIIoT-ML Dataset

In Figure 27 it's noticeable that, just like on the previous dataset, the difference between balancing the data using limitation or Smote is negligible. This most-likely excludes the possibility that the model is being impacted by less data since both methods of balancing produce the same results, even though one has considerably more information than the other. This also means that, in the case of this model and dataset, there is no reason to go for a Smote balancing, which takes considerably more time, for no improvement, as seen in Figure 28.

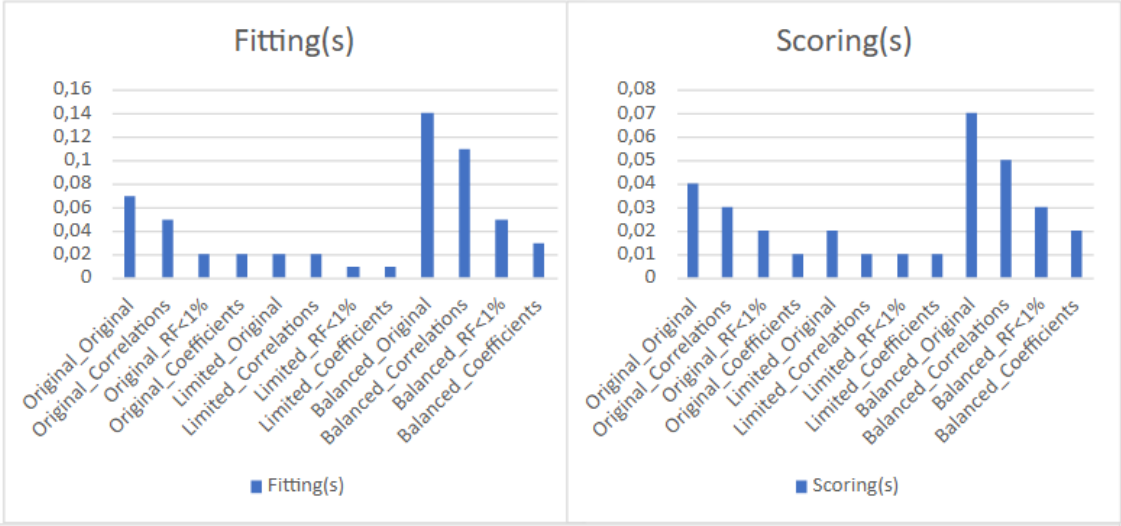


Figure 28: Fitting and Scoring graph of the GNB results on the EdgeIIoT-ML Dataset

6.2.2.3 Random Forest

Table 22: Table of the Random Forest results on the EdgeIIoT-ML Dataset

RandomForest						
Features	Accuracy	Precision	Recall	F1	Fitting(s)	Scoring(s)
Original_Original	99,52%	99,13%	99,01%	99,07%	1,54	0,14
Original_Correlations	99,30%	98,73%	98,57%	98,65%	1,52	0,65
Original_RF<1%	99,49%	99,11%	98,93%	99,02%	1,12	0,12
Original_Coefficients	99,49%	99,13%	98,93%	99,03%	1,12	0,13
Limited_Original	99,22%	99,22%	99,22%	99,22%	0,65	0,1
Limited_Correlations	99,01%	99,01%	99,01%	99,01%	0,64	0,1
Limited_RF<1%	99,11%	99,11%	99,11%	99,11%	0,45	0,1
Limited_Coefficients	99,15%	99,15%	99,15%	99,15%	0,47	0,1
Balanced_Original	99,52%	99,52%	99,52%	99,52%	3,01	0,19
Balanced_Correlations	99,34%	99,35%	99,34%	99,34%	3,16	0,19
Balanced_RF<1%	99,42%	99,42%	99,42%	99,42%	2,89	0,18
Balanced_Coefficients	99,44%	99,44%	99,44%	99,44%	2,86	0,18

The Random Forest model is, once again, the one with the best performance, with no metric going under 98%, and only 5 values under 99%, as seen in Table 22. The performance is very close between all distributions and feature removal techniques, as seen in Figure 29.

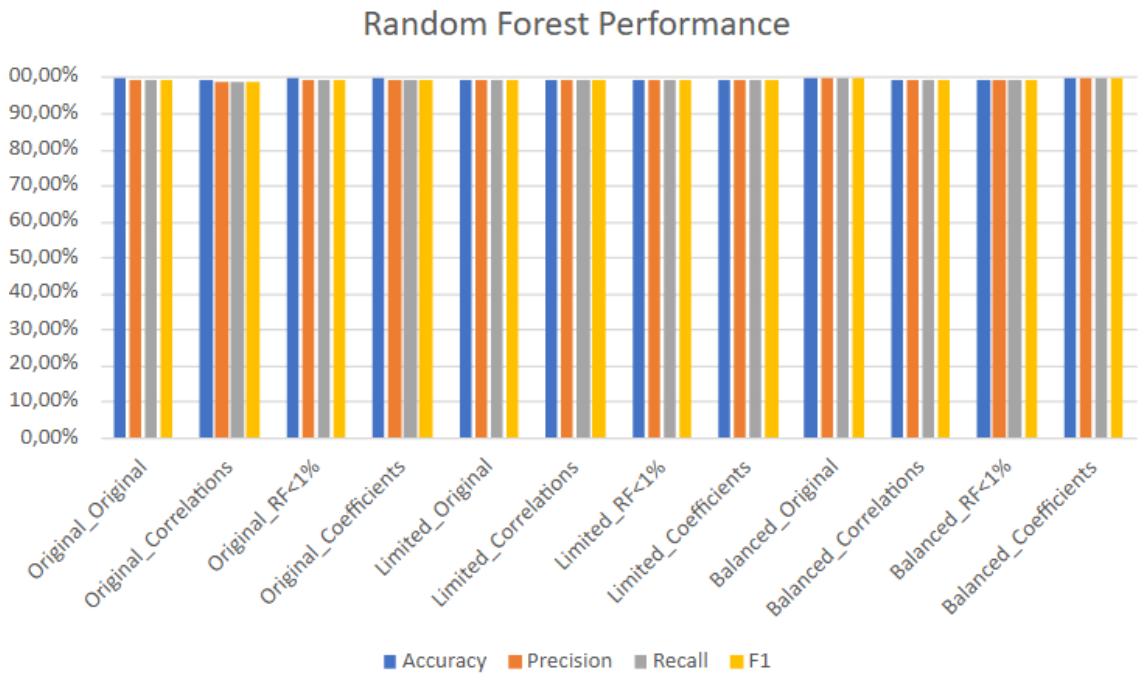


Figure 29: Graph of the Random Forest results on the EdgeIIoT-ML Dataset

At first glance, it seems that every model is the same, however, upon rescaling the graph, it is possible to see the differences across the different conditions.

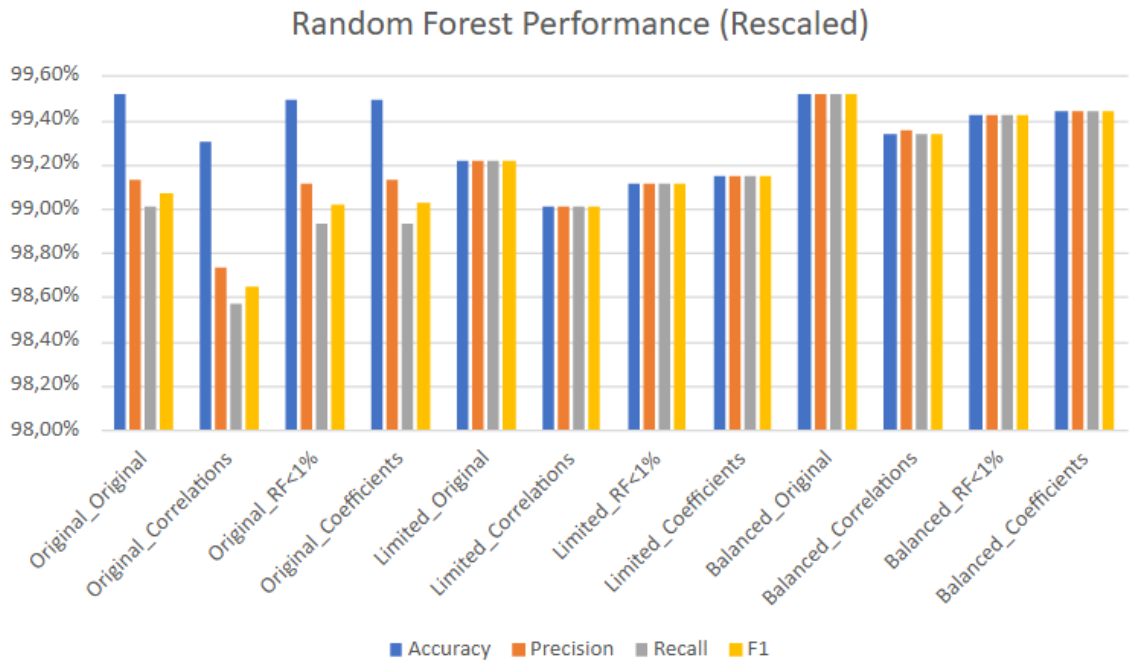


Figure 30: Rescaled graph of the Random Forest results on the EdgeIIoT-ML Dataset

Figure 30 once again shows the same consistency across metrics in the balanced results, and, once again, the Smote balanced data showed better results. Looking at the time taken to fit and score the data, Figure 31, it is also noticeable that the

scoring time is not much higher than the original distribution, with the model simply taking more time to fit.

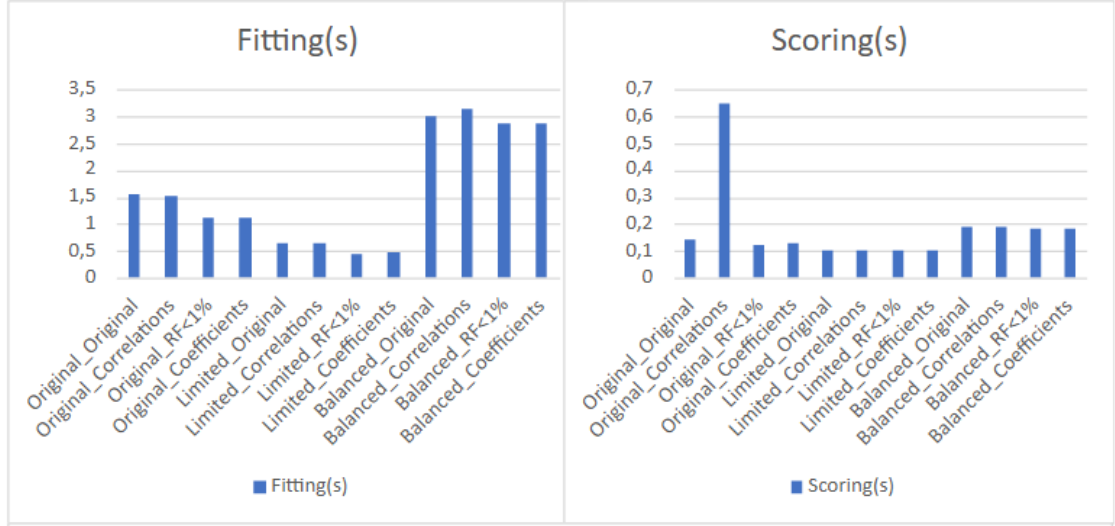


Figure 31: Fitting and Scoring graph of the Random Forest results on the EdgeIIoT-ML Dataset

6.2.3 EdgeIIoT-DNN Dataset

This is the other dataset that is present in the EdgeIIoT package. This dataset has around 7 times the amount of data that the Classical Machine Learning dedicated one has, due to the high demand of information Neural Networks typically have. Figure 32 shows the original data distribution of the dataset.

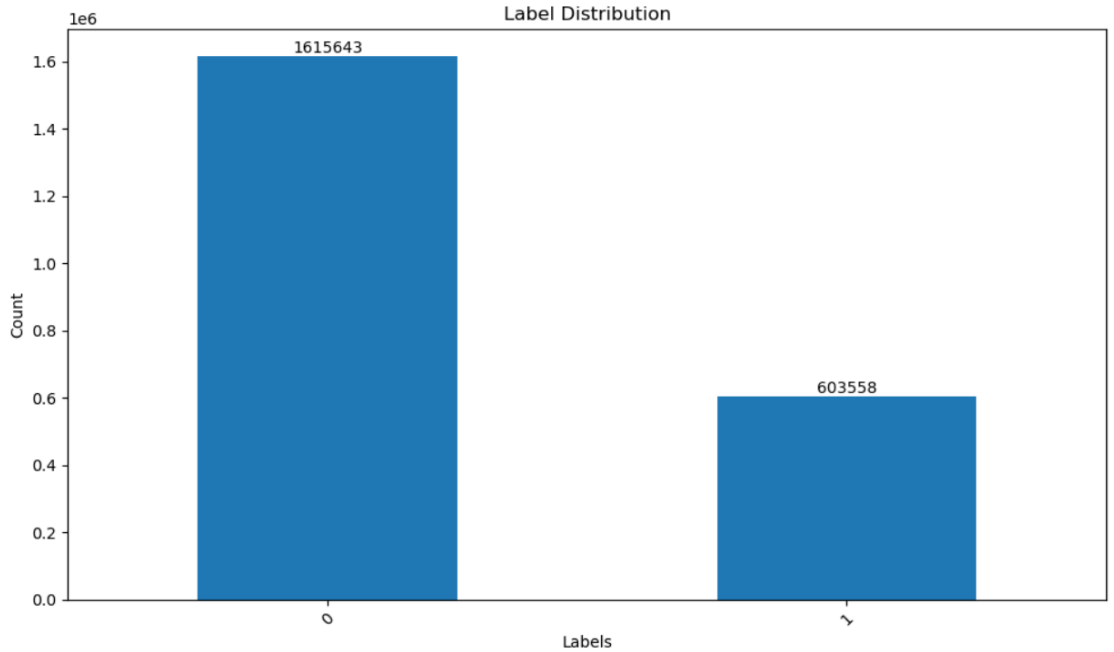


Figure 32: Original data distribution of the EdgeIIoT-DNN Dataset

Once again, the limited approach will reduce the number of normal traffic to 603558, and the Smote approach will increase the number of malicious traffic to 1615643.

6.2.3.1 KNN

Table 23: Table of the KNN results on the EdgeIIoT-DNN Dataset

KNN						
Features	Accuracy	Precision	Recall	F1	Fitting(s)	Scoring(s)
Original_Original	94,70%	95,52%	91,04%	92,97%	0,41	511,32
Original_Correlations	94,69%	95,51%	91,03%	92,96%	0,28	432,57
Original_RF<1%	94,67%	95,50%	91,00%	92,94%	4,94	16,23
Original_Coefficients	93,46%	94,48%	88,93%	91,23%	3,28	15,76
Limited_Original	88,14%	88,18%	88,14%	88,14%	0,21	156,49
Limited_Correlations	88,10%	88,13%	88,10%	88,10%	0,14	130,03
Limited_RF<1%	88,12%	88,16%	88,12%	88,12%	2,67	9,75
Limited_Coefficients	86,01%	86,05%	86,01%	86,00%	1,75	10,29
Balanced_Original	95,33%	95,47%	95,33%	95,33%	0,91	1109,67
Balanced_Correlations	95,35%	95,48%	95,35%	95,34%	0,53	977,92
Balanced_RF<1%	95,32%	95,45%	95,32%	95,31%	7,94	25,75
Balanced_Coefficients	94,46%	94,62%	94,46%	94,46%	5,27	33,88

In Table 23 we can see that the KNN model got a substantial boost in performance when compared to the Classical Machine Learning version of the dataset.

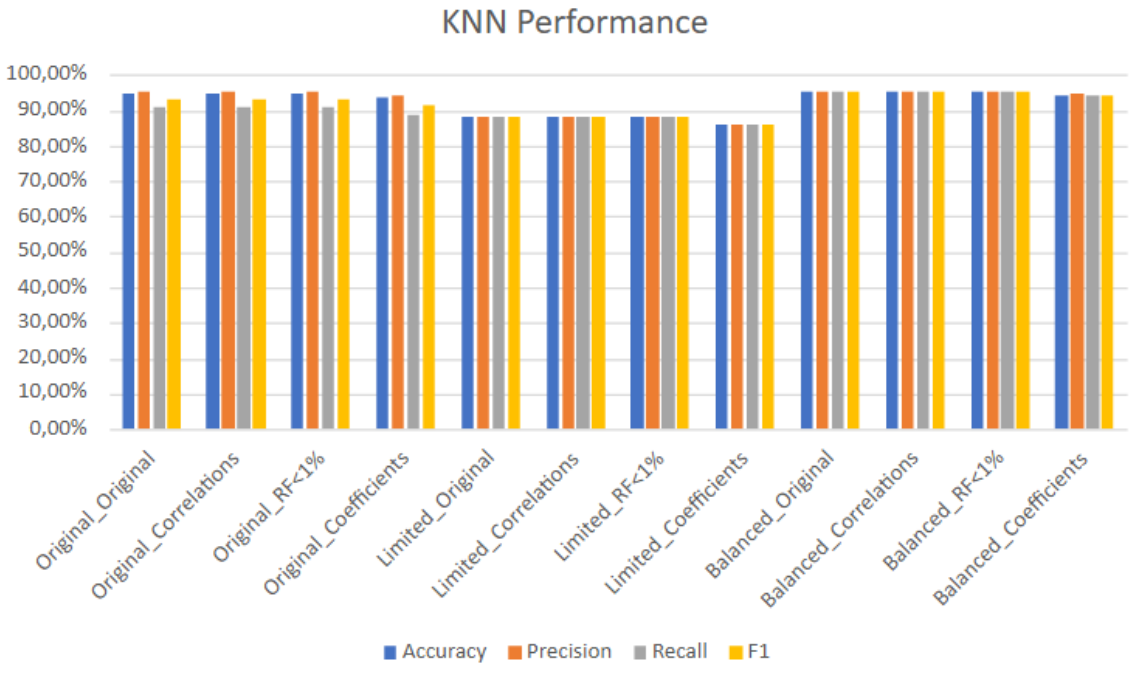


Figure 33: Graph of the KNN results on the EdgeIIoT-DNN Dataset

Figure 33 shows that, once again, the performance of the data that was balanced with Smote is better than the original distribution.

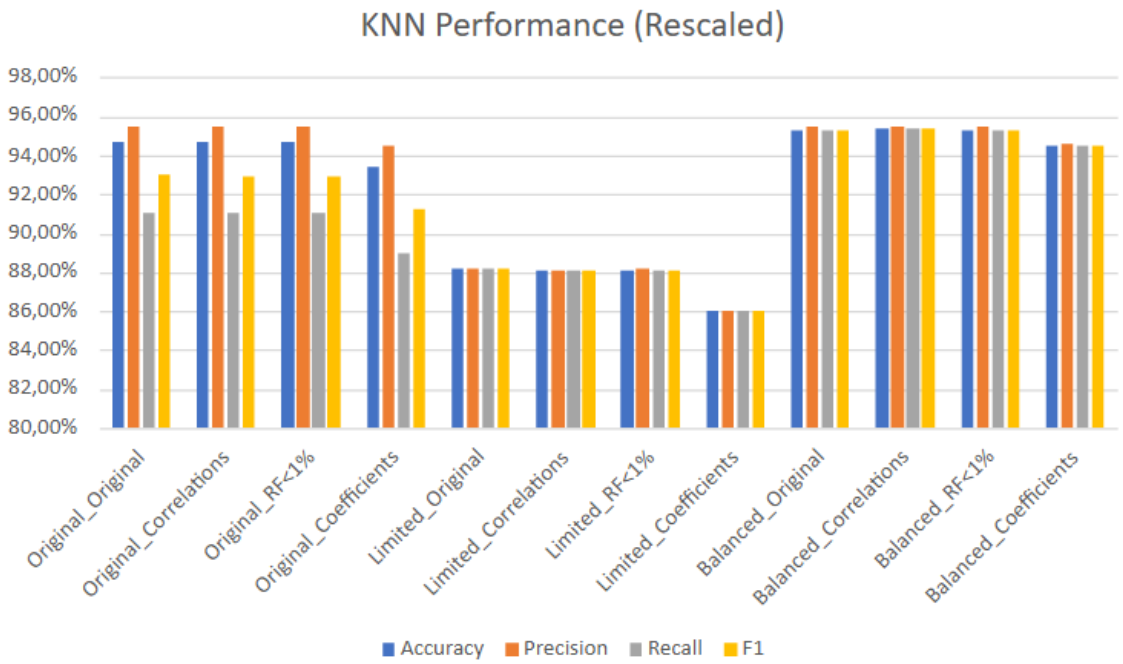


Figure 34: Rescaled graph of the KNN results on the EdgeIIoT-DNN Dataset

When re-scaling the graphic to better show the fluctuations, as seen in Figure 34, we can not only see the usual stabilization of the metrics, but also that, in this case, the Coefficients approach lost a lost of performance, especially on the limited data.

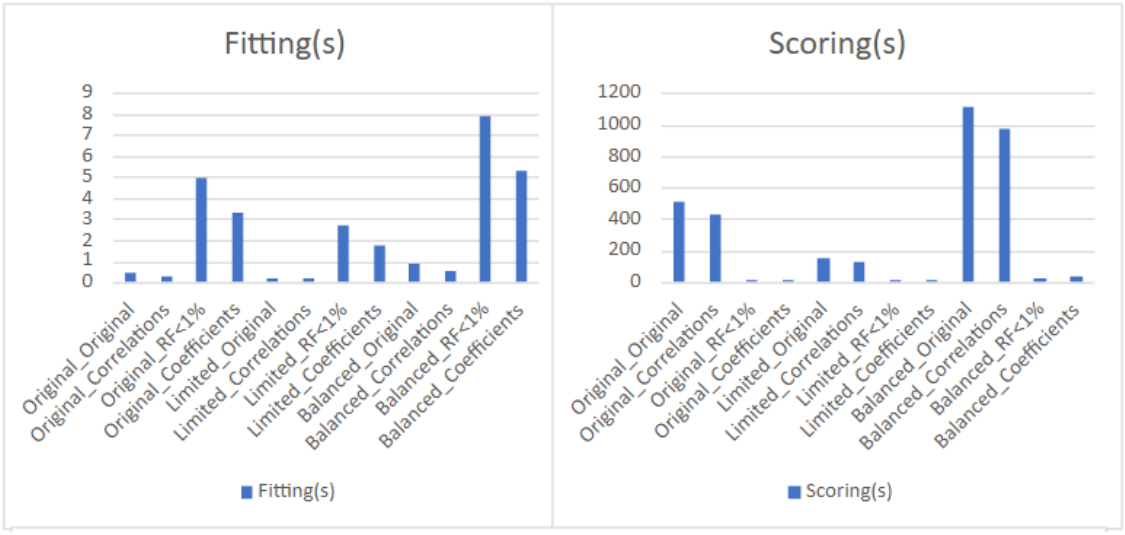


Figure 35: Fitting and Scoring graph of the KNN results on the EdgeIIoT-DNN Dataset

Although the performance of the model increased, Figure 35 shows that the time taken to score the data got ridiculously high, taking 100 times the time that was

needed in the Classical Machine Learning dataset. It is, however, also where we see a great result for the feature elimination techniques, with the Coefficients approach reducing the time to score the data from 1109 to 25 seconds while keeping the same performance on the data that was balanced with Smote.

6.2.3.2 GNB

Table 24: Table of the GNB results on the EdgeIIoT-DNN Dataset

GNB						
Features	Accuracy	Precision	Recall	F1	Fitting(s)	Scoring(s)
Original_Original	78,17%	80,05%	61,57%	62,64%	0,92	0,47
Original_Correlations	78,18%	80,06%	61,58%	62,65%	0,59	0,33
Original_RF<1%	77,68%	76,85%	61,53%	62,41%	0,36	0,24
Original_Coefficients	78,04%	88,41%	59,62%	59,59%	0,28	0,2
Limited_Original	61,54%	74,37%	61,54%	55,71%	0,51	0,26
Limited_Correlations	61,51%	74,25%	61,51%	55,69%	0,32	0,18
Limited_RF<1%	60,71%	71,38%	60,70%	55,22%	0,19	0,12
Limited_Coefficients	59,67%	77,68%	59,67%	51,84%	0,16	0,1
Balanced_Original	61,50%	74,28%	61,50%	55,66%	1,59	0,79
Balanced_Correlations	61,51%	74,30%	61,51%	55,68%	1,01	0,56
Balanced_RF<1%	60,38%	72,08%	60,38%	54,32%	0,59	0,38
Balanced_Coefficients	59,64%	77,67%	59,64%	51,79%	0,46	0,32

Table 24 shows that there was a slight improvement in the model, especially on the original distribution using the original features set as well as using the correlations set.

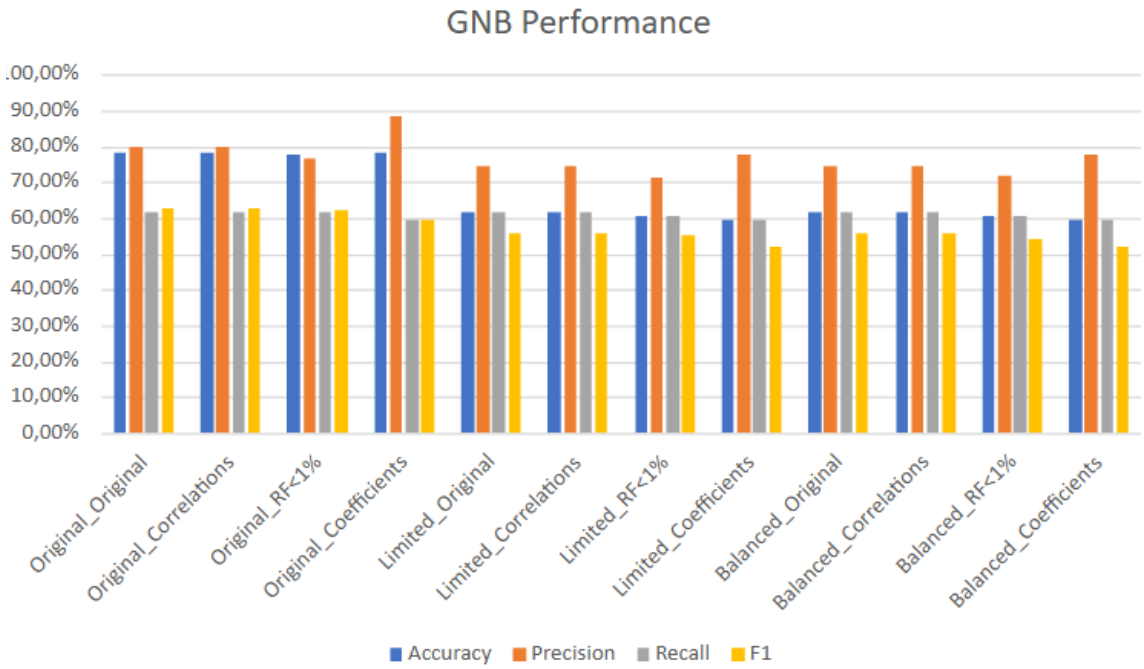


Figure 36: Graph of the GNB results on the EdgeIIoT-DNN Dataset

Looking at Figure 36, however, once again shows that the model does not improve much when using more information, especially if the data is balanced. This is because the results of both the limited and balanced tests were basically the same.

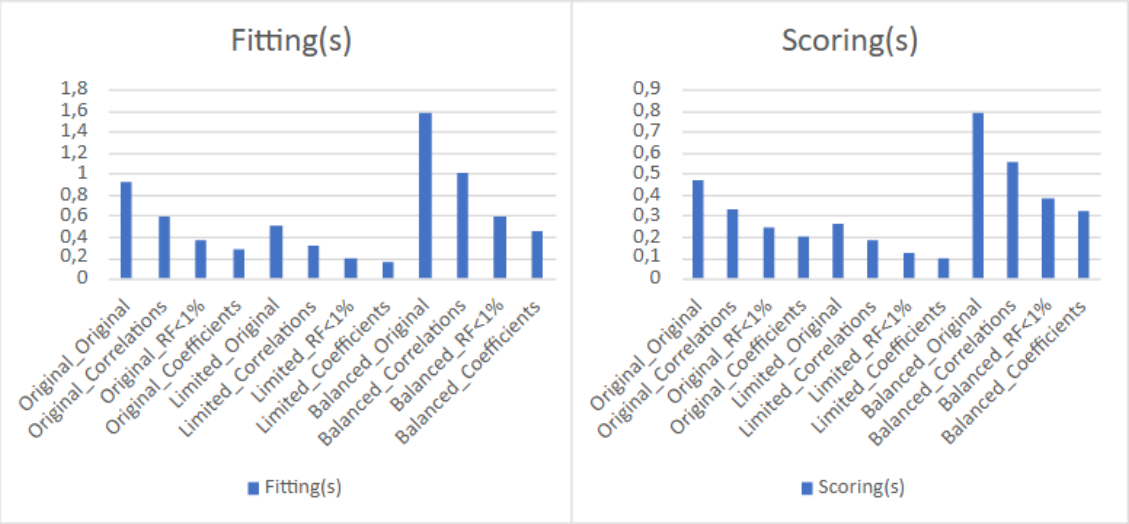


Figure 37: Fitting and Scoring graph of the GNB results on the EdgeIIoT-DNN Dataset

Looking at the times shown in Figure 37, it is also seen that they follow the same pattern as the previous dataset, however, the fitting time has scaled together with the amount of data.

6.2.3.3 Random Forest

Table 25: Table of the Random Forest results on the EdgeIIoT-DNN Dataset

RandomForest						
Features	Accuracy	Precision	Recall	F1	Fitting(s)	Scoring(s)
Original_Original	99,53%	99,65%	99,16%	99,41%	26,11	0,97
Original_Correlations	99,53%	99,65%	99,17%	99,41%	26,57	0,93
Original_RF<1%	99,48%	99,63%	99,05%	99,34%	25	0,92
Original_Coefficients	98,59%	99,00%	97,46%	98,19%	29,69	1,08
Limited_Original	99,31%	99,31%	99,31%	99,31%	14,81	0,52
Limited_Correlations	99,32%	99,32%	99,32%	99,32%	14,43	0,5
Limited_RF<1%	99,29%	99,29%	99,29%	99,29%	13,15	0,48
Limited_Coefficients	97,69%	97,70%	97,69%	97,69%	15,79	0,58
Balanced_Original	99,57%	99,57%	99,57%	99,57%	45,54	1,57
Balanced_Correlations	99,57%	99,57%	99,57%	99,57%	47	1,52
Balanced_RF<1%	99,56%	99,56%	99,56%	99,56%	43,13	1,46
Balanced_Coefficients	98,69%	98,69%	98,69%	98,69%	49,01	1,68

We once again see the Random Forest model as the best performing model, as shown in Table 25. The performance of the model also did go up, especially on the original distribution, where there was more space for improvement.

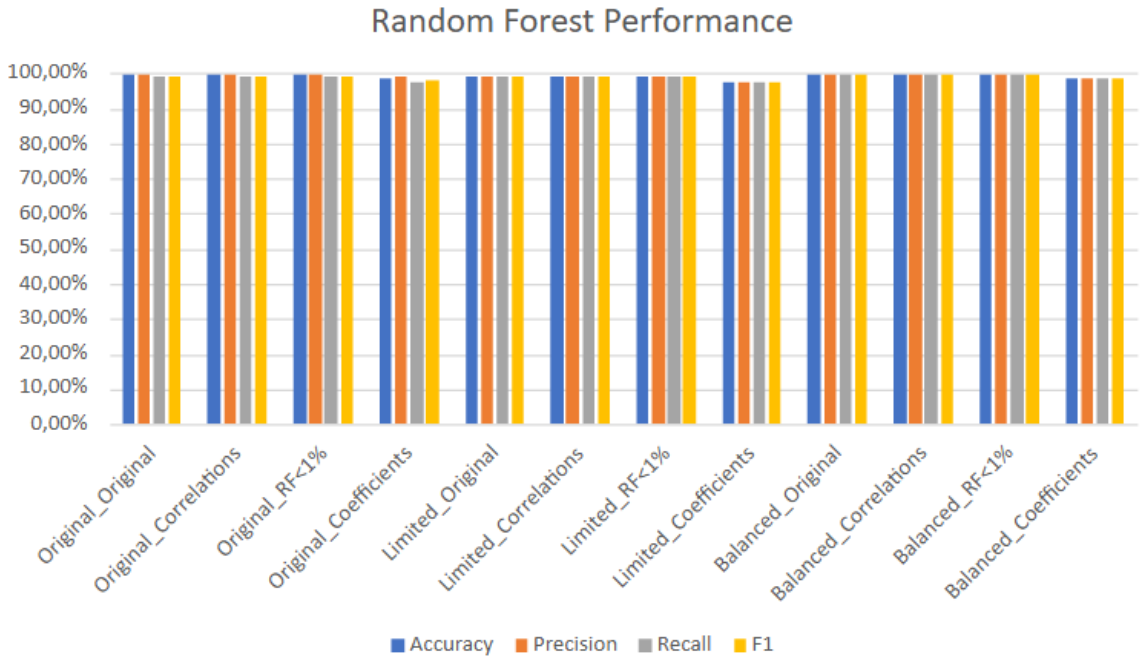


Figure 38: Graph of the Random Forest results on the EdgeIIoT-DNN Dataset

However, as seen on Figure 38, there is a slight loss of performance on the bal-

anced versions of the data when using the correlations, which can better be seen in the re-scaled version shown in Figure 39.

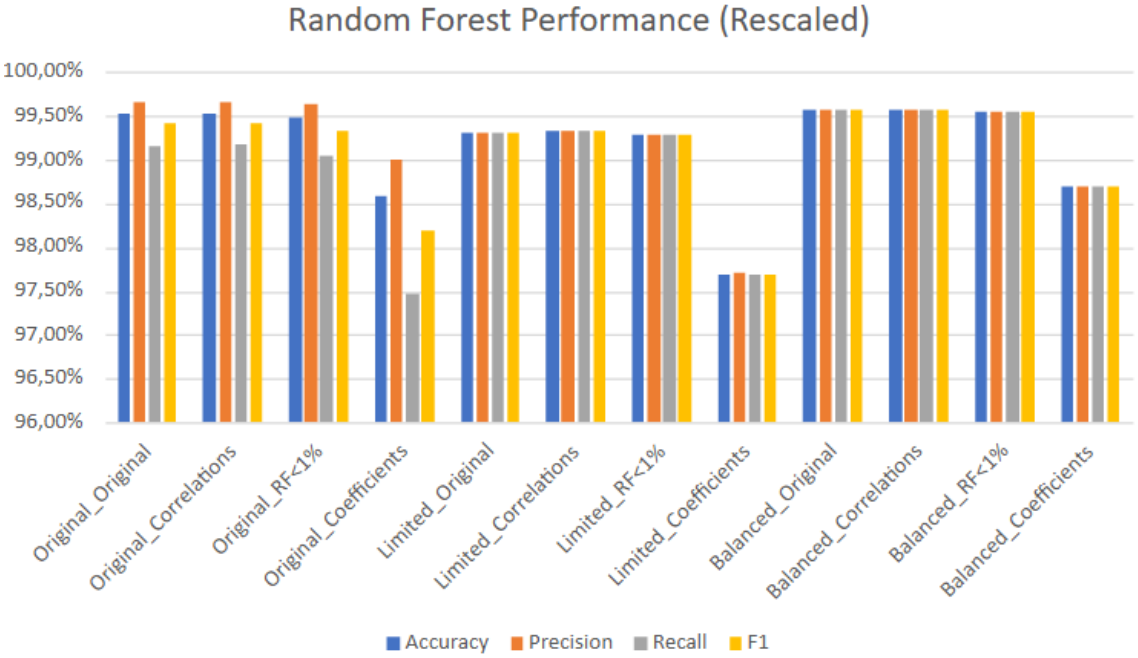


Figure 39: Rescaled graph of the Random Forest results on the EdgeIIoT-DNN Dataset

The time to both fit and score the data also went up significantly, as seen in Figure 40, which might not be worth the performance gained.

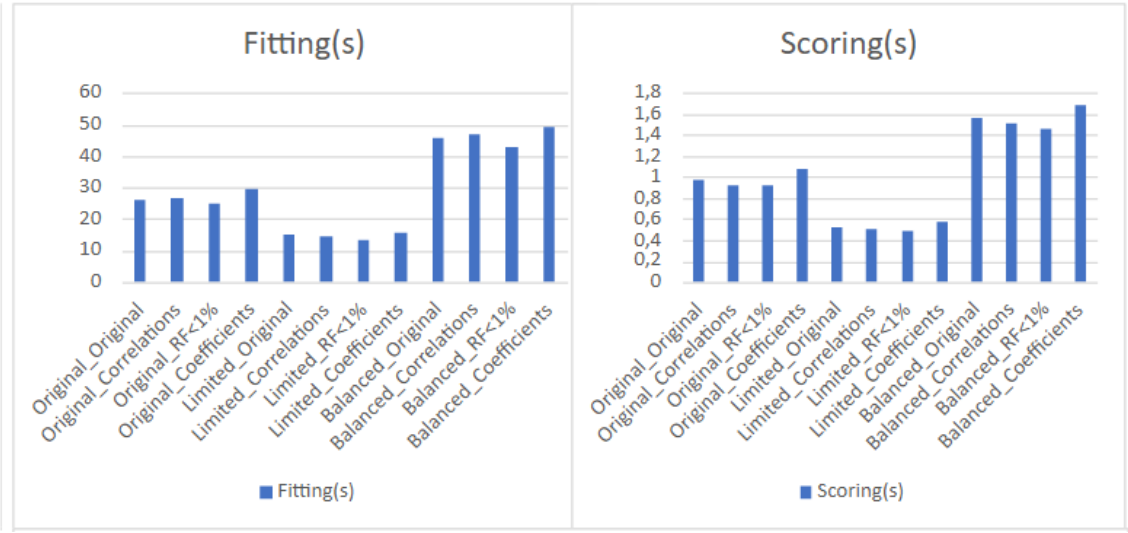


Figure 40: Fitting and Scoring graph of the Random Forest results on the EdgeIIoT-DNN Dataset

6.2.4 UCDataset

This is the dataset that was used to test the performance of the model using metrics that were not only network-related. This dataset is not labelled in a similar way to the others. Instead of only 'Normal' and 'Attack' identification, it has a 'pre', 'attack' and 'pos' labeling, which allows us to check its ability to predict attacks before they happen, by checking for, for example, smaller shifts in usual performance. Figure 41 shows the original distribution of the dataset.

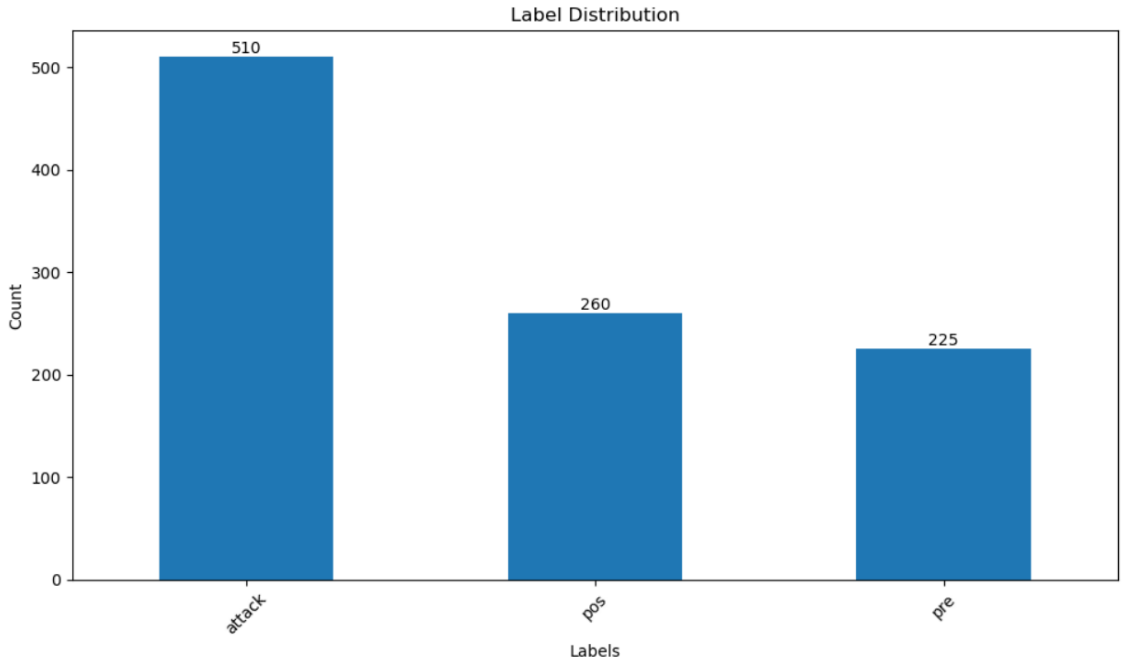
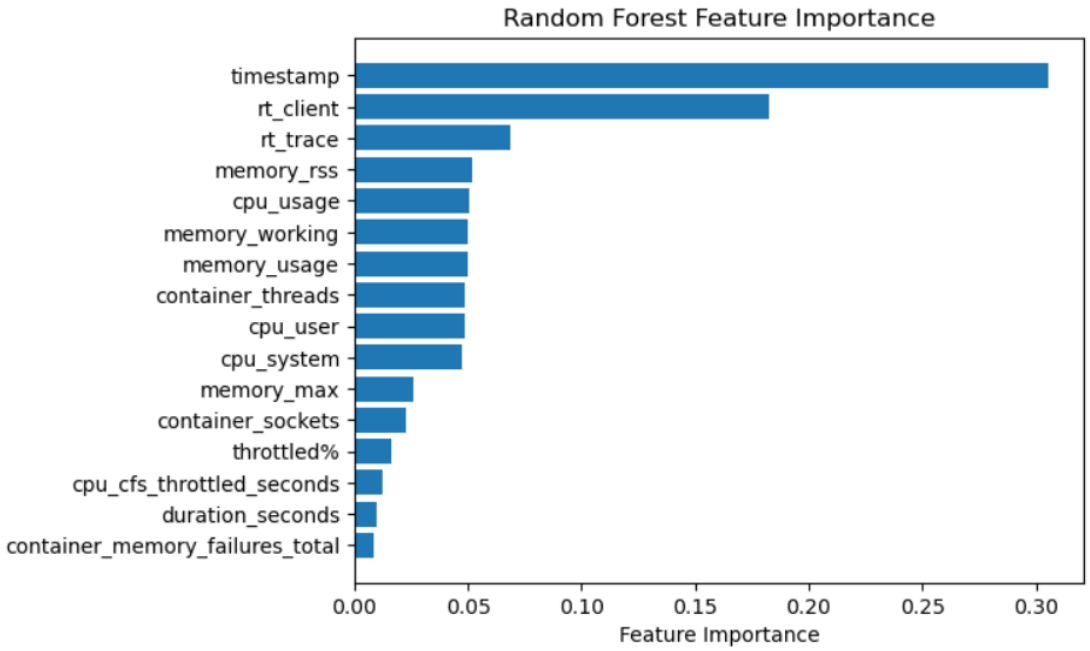


Figure 41: Original data distribution of the UC Dataset

During the first tests with this dataset, the models were giving very high importance to the feature 'timestamp', which did not make much sense, as this feature had nothing to do with the performance of the environment. Upon further investigation, the problem was found. This dataset must have been done automatically, and there was no randomness introduced and, as such, all of the attacks occurred with a consistent time between them. The model was picking up on this, and was using it to identify the attacks instead of the system features, as seen in Figure 42.




```
timestamp: 30.555363816435776
rt_client: 18.271252156462136
rt_trace: 6.863580998433974
memory_rss: 5.186790629073265
cpu_usage: 5.067764231029136
memory_working: 4.984131203666985
memory_usage: 4.966088385050864
container_threads: 4.8793392577015755
cpu_user: 4.85779445505788
cpu_system: 4.758304573570843
memory_max: 2.6069543049978843
container_sockets: 2.2738150723723765
throttled%: 1.6391818938823084
cpu_cfs_throttled_seconds: 1.266435046306497
duration_seconds: 0.9889729440741982
container_memory_failures_total: 0.8342310318842959
```

	timestamp	container_memory_failures_total	container_sockets	container_threads	throttled%
pre	1.000000	0.000008	0.000000	0.000000	0.000000
attack	0.999994	0.000015	0.000000	0.000000	0.000000
pos	0.999993	0.000001	0.000000	0.000000	0.000000

Figure 42: Screenshot of feature importance outputs

To prevent this, this feature was removed, which hindered the performance of the models, but this performance was obtained in a non-realistic scenario.

6.2.4.1 KNN

Table 26: Table of the KNN results on the UC Dataset

KNN						
Features	Accuracy	Precision	Recall	F1	Fitting(s)	Scoring(s)
Original_Original	72,76%	69,82%	65,78%	66,78%	0,0019	0,1873
Original_Correlations	67,95%	61,45%	60,17%	60,29%	0,0016	0,1989
Original_RF<1%	70,44%	66,94%	63,05%	63,99%	0,0015	0,1924
Original_Coefficients	72,76%	69,82%	65,78%	66,78%	0,0017	0,1861
Limited_Original	63,24%	63,12%	63,38%	62,58%	0,0018	0,1976
Limited_Correlations	62,39%	61,84%	62,51%	61,72%	0,0014	0,1876
Limited_RF<1%	62,22%	62,76%	62,31%	61,43%	0,0015	0,1807
Limited_Coefficients	60,63%	61,19%	60,91%	60,15%	0,0012	0,181
Balanced_Original	74,71%	74,76%	74,75%	74,54%	0,0027	0,2057
Balanced_Correlations	73,92%	74,15%	74,10%	73,84%	0,0017	0,2057
Balanced_RF<1%	75,03%	75,33%	75,20%	74,93%	0,0023	0,1828
Balanced_Coefficients	76,67%	76,73%	76,85%	76,67%	0,002	0,1887

When looking at Table 26, the performance of the KNN model is slightly worse than what was shown with previous datasets, however, this could be because this dataset consists of 1000 lines while the others are in the hundreds of thousands, or even in the millions.

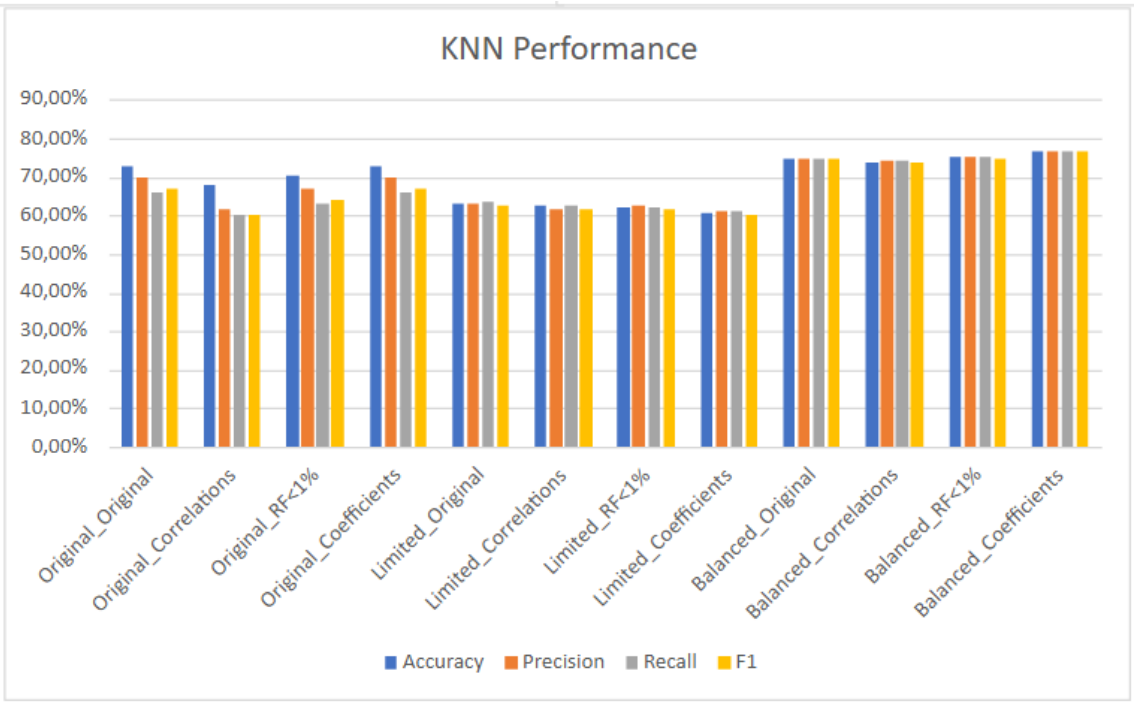


Figure 43: Graph of the KNN results on the UC Dataset

The same behavior, however, can be seen in Figure 43, where after balancing the data the results became more consistent, and the performance was improved when balancing using Smote.

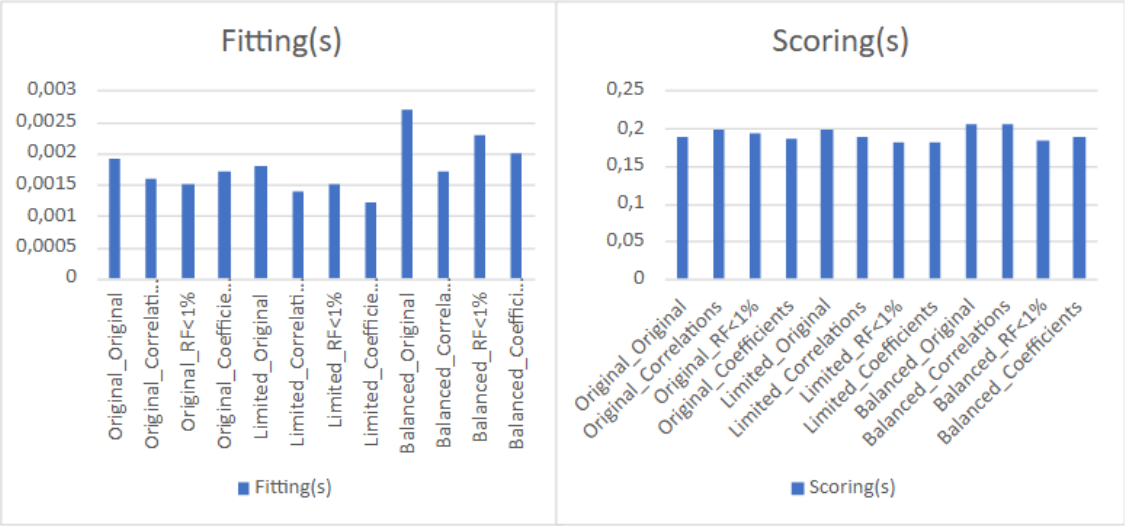


Figure 44: Fitting and Scoring graph of the KNN results on the UC Dataset

The time taken to fit and score the data, shown in Figure 44 also goes accordingly to what was seen, with the balanced data taking more time to train, but producing better results.

6.2.4.2 GNB

Table 27: Table of the GNB results on the UC Dataset

GNB						
Features	Accuracy	Precision	Recall	F1	Fitting(s)	Scoring(s)
Original_Original	55,48%	49,48%	53,50%	43,43%	0,0015	0,0045
Original_Correlations	37,49%	45,40%	42,06%	30,37%	0,001	0,0052
Original_RF<1%	55,78%	50,03%	53,21%	42,83%	0,0013	0,0045
Original_Coefficients	55,58%	49,48%	53,50%	43,43%	0,0012	0,0045
Limited_Original	48,00%	47,12%	47,86%	39,56%	0,0012	0,0044
Limited_Correlations	41,78%	44,95%	41,59%	32,44%	0,0013	0,0047
Limited_RF<1%	51,99%	47,54%	52,11%	43,00%	0,0013	0,0046
Limited_Coefficients	51,41%	44,77%	51,38%	42,18%	0,0013	0,0042
Balanced_Original	50,07%	60,40%	50,00%	42,56%	0,0019	0,0052
Balanced_Correlations	43,46%	57,96%	43,38%	36,14%	0,0017	0,0046
Balanced_RF<1%	53,07%	58,51%	53,13%	44,03%	0,0017	0,005
Balanced_Coefficients	46,21%	61,52%	46,16%	39,21%	0,0018	0,0048

The results of the GNB model, shown in Table 27, also go accordingly with the previous results, with an inferior performance and very inconsistent results, which can be seen easily in Figure 45.

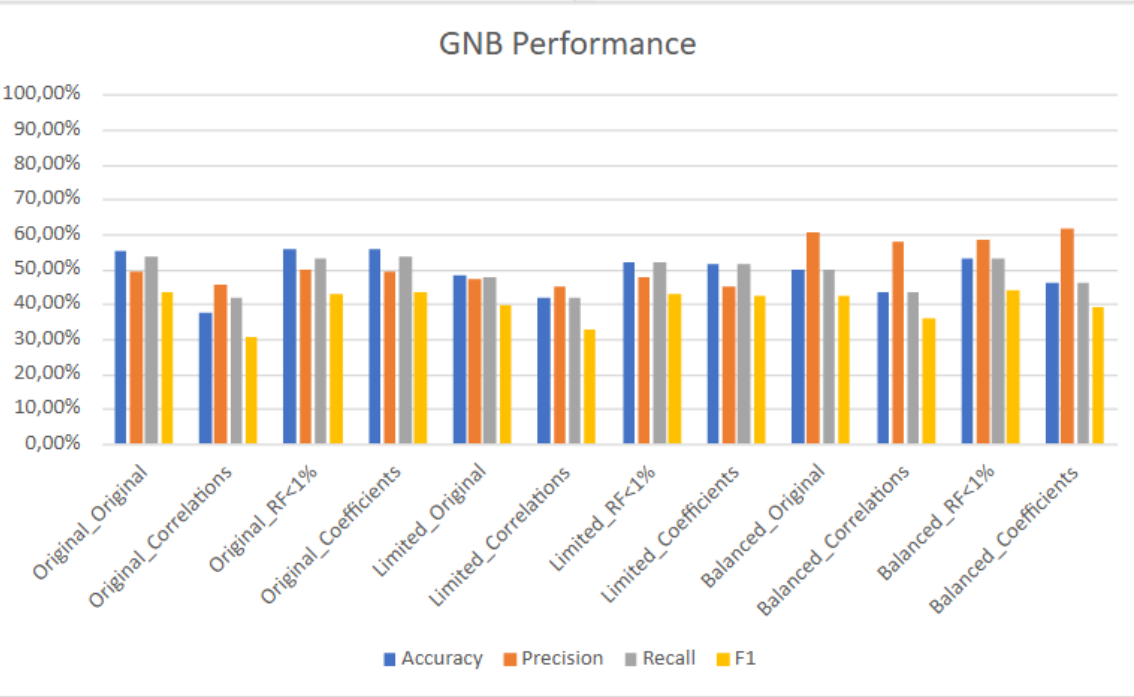


Figure 45: Graph of the GNB results on the UC Dataset

As for the time to fit and score the data, seen in Figure 46, once again the balanced model takes more time to train, however, the scoring time stayed pretty consistent across all conditions.

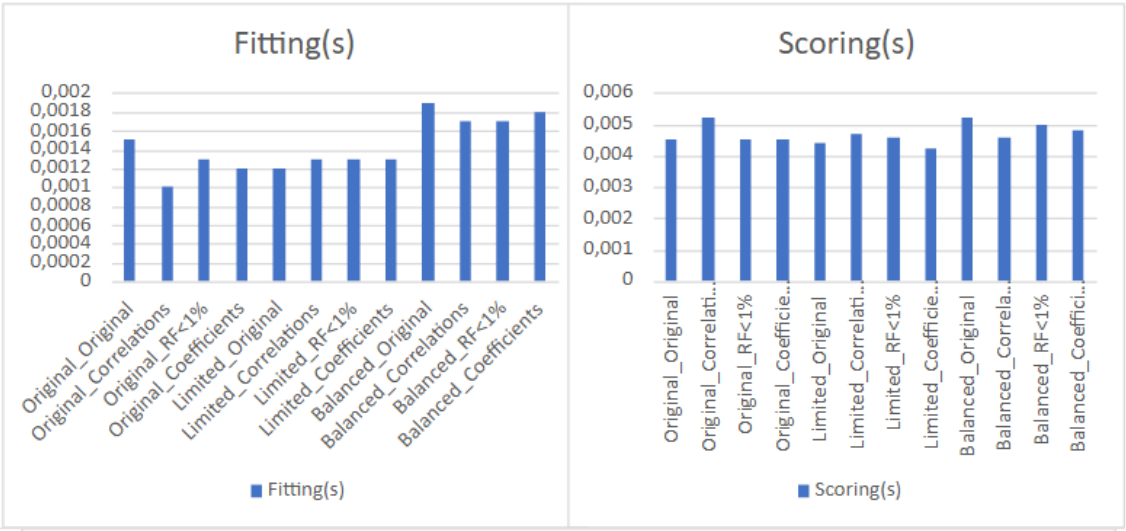


Figure 46: Fitting and Scoring graph of the GNB results on the UC Dataset

6.2.4.3 Random Forest

Table 28: Table of the Random Forest results on the UC Dataset

RandomForest						
Features	Accuracy	Precision	Recall	F1	Fitting(s)	Scoring(s)
Original_Original	94,58%	94,72%	93,98%	94,22%	0,1058	0,0555
Original_Correlations	65,93%	57,16%	56,16%	55,84%	0,0968	0,0517
Original_RF<1%	93,98%	93,68%	93,53%	93,47%	0,103	0,0543
Original_Coefficients	74,07%	71,29%	67,23%	68,10%	0,0956	0,059
Limited_Original	93,04%	93,20%	93,07%	92,96%	0,0985	0,0606
Limited_Correlations	58,08%	58,34%	57,84%	57,71%	0,0992	0,0568
Limited_RF<1%	93,94%	93,18%	92,98%	92,88%	0,0932	0,0565
Limited_Coefficients	68,32%	68,84%	68,64%	68,18%	0,1041	0,0599
Balanced_Original	95,23%	95,21%	96,26%	95,18%	0,1136	0,0674
Balanced_Correlations	78,17%	78,55%	78,36%	78,18%	0,1103	0,0594
Balanced_RF<1%	94,77%	94,78%	94,84%	94,76%	0,1074	0,0571
Balanced_Coefficients	81,44%	81,60%	81,59%	81,33%	0,104	0,0578

Finally, the Random Forest model keeps the crown as the best performance, however, this time the Correlations and Coefficients approach get a heavy hit in performance, which can be easily seen in Figure 47.

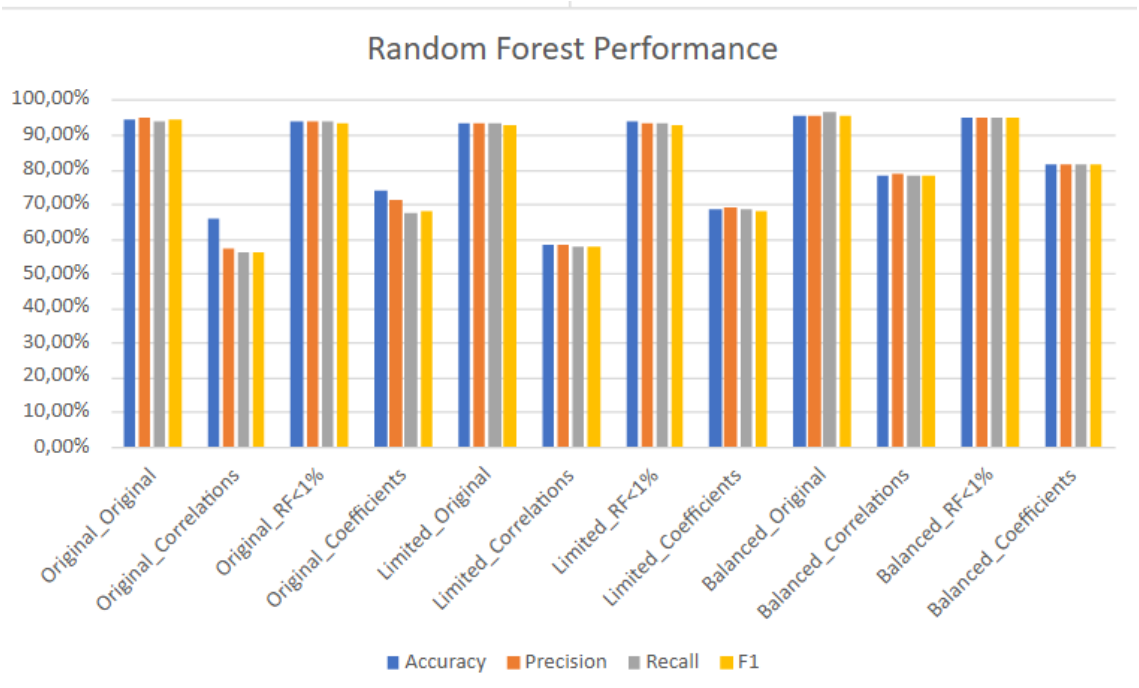


Figure 47: Graph of the Random Forest results on the UC Dataset

This is probably because these two approaches remove features that the Random Forest deems as important, however, as this is a much smaller dataset, it cannot compensate with other information, as it doesn't have enough to do so.

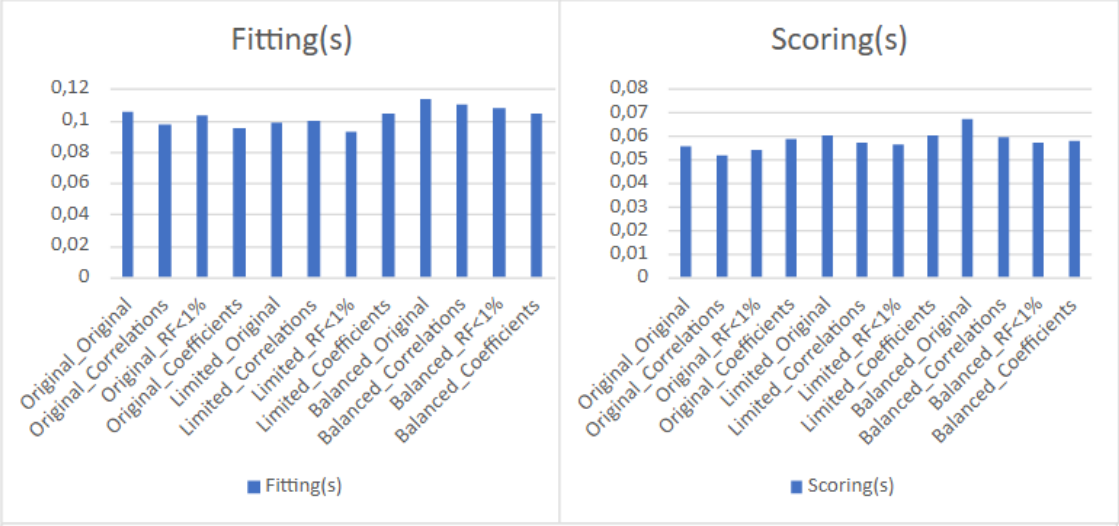


Figure 48: Fitting and Scoring graph of the Random Forest results on the UC Dataset

As for the time taken to score and fit, shown in Figure 48, it also is pretty consistent, especially because all of the values are extremely small, once again because the dataset is very small.

6.3 Discussion

As shown throughout the previous section, there are several improvements made across the models and datasets. To better summarize the boosts of performance, Table 29 shows the average difference in performance of the KNN model.

Table 29: Improvements/Loss of performance of the KNN model for each approach

Approach	Δ(Metric Average)	Δ(Fitting Time Average)	Δ(Scoring Time Average)
Limiting	-5,57%	489,93%	-33,58%
Smote Balancing	4,12%	332,91%	-45,72%
Random Forest	-0,48%	351,45%	-35,25%
Correlations	-0,75%	456,91%	-28,05%
Coefficients	-0,14%	466,64%	-58,15%

The average scoring time shows that all of the approaches result in better results from a time perspective, with the tradeoff of some classification performance and

a bigger fitting time. For the Smote balancing, however, there is also a boost in classification performance, showing that, on average, the model ends up with metrics 4% higher.

Table 30: Improvements/Loss of performance of the GNB model for each approach

Approach	Δ(Metric Average)	Δ(Fitting Time Average)	Δ(Scoring Time Average)
Limiting	-0,73%	-27,88%	-29,67%
Smote Balancing	0,13%	-38,57%	-32,50%
Random Forest	2,74%	-35,35%	-27,20%
Correlations	-3,60%	-25,90%	-22,68%
Coefficients	0,56%	-47,53%	-40,64%

Table 30 also shows an improvement in all of the time related values, as well as bigger averages for the metrics, with the exception of the Limiting approach, which would be expected, and the Correlations approach.

Table 31: Improvements/Loss of performance of the Random Forest model for each approach

Approach	Δ(Metric Average)	Δ(Fitting Time Average)	Δ(Scoring Time Average)
Limiting	-1,65%	-12,03%	-1,70%
Smote Balancing	2,62%	-6,09%	-1,15%
Random Forest	7,35%	-3,39%	-3,92%
Correlations	-8,07%	-13,50%	30,23%
Coefficients	-4,92%	-12,32%	-0,85%

Finally, in Table 31, we can see that, again, most of the approaches result in smaller fitting and scoring times, with the best approach being Random Forest, with a 7% increase in performance, which was expected since the values used for this approach were extracted directly from the model itself. It is important to note that, although this value is big, it is very likely influence of the enormous Random Forest improvements on the UCDataset, as on all of the other datasets this models was close to 99%, giving no space for a 7% increase.

Despite all of the corrections made from the first method, these results also have its limitations, although these are not as method-related. The following are all of the identified limitations.

- **Custom Dataset**: One of the objectives of this work was to test the models with data that was extracted from a controlled environment, providing a dataset with features that we were absolutely sure that could be extracted. The system to extract this data was being developed alongside the work

done for this thesis, however, it was not ready in time, which prevented the models from being tested with that dataset, which was "replaced" by UCDataset as a way to still test the model in similar conditions.

- **Attack identification**: As mentioned before, all of the tests were done using binary identification of the attacks. Although this improved the performance of the models, there are cases where this type of identification would not be enough, but testing the models with this type of identification would require a lot more time, especially to balance the data with Smote, which would increase the total lines of the datasets to 8 digits.

6.4 Summary

In this chapter, the results of the second method were presented. First, the new dataset that was introduced was addressed, followed by the results of the second model, which were divided into datasets and models. After that, the conclusion of those results was presented as well as the limitations that apply to that work.

Chapter 7

Conclusion

In conclusion, the model with the best overall performance is Random Forest, which, even with the original distribution and all features, can consistently score upwards of 98% in all metrics. The other models, however, also show great results, especially with the time taken to fit and score the data. The time to achieve these scores, however, can be heavily reduced with the techniques shown throughout this thesis.

Balancing the dataset by limiting the amount of data to match the smallest sample can drastically increase the speed at which the models are trained and how long they take to predict, costing a fraction of the performance they provide. This, in cases where speed is more important than perfect accuracy, can be quite significant.

If, however, the goal is to squeeze the most performance possible out of the models, using Smote to balance the dataset can be a great solution, but this will increase the time required by the models to do their work, although this can be countered by using the feature selection approaches as well, as shown in the discussion of Chapter 6.

One other great option is to binarily identify the attacks. That is, instead of identifying exactly what attack is being made, label them only as Attack and Normal. This can both speed up the model and increase its accuracy. This can be a great solution if the goal of the models is to only detect if an environment is under attack without needing to specify which one, leaving that part to the admins.

The choice to be made will always depend on the main goal that is set for the models.

As for the next steps of this work, one is to write and publish a paper that summarizes all of the information obtained during this thesis. As the feature extraction mechanism advances, it will also be possible to use it together with the models, as planned in the beginning. Following this line, a Remote Behaviour Monitoring system is also being developed, where the models will be housed when a final implementation is developed, so making sure they are compatible with this system should also be done in the future.

References

Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing v4.0. *Cloud Security Alliance*, 4, 2017. ISSN ? doi: ?

Saud Alzughaibi and Salim El Khediri. A cloud intrusion detection systems based on dnn using backpropagation and pso on the cse-cic-ids2018 dataset. *Applied Sciences*, 13(4), 2023. ISSN 2076-3417. doi: 10.3390/app13042276. URL https://www.mdpi.com/2076-3417/13/4/2276.

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing, 2010. ISSN 00010782.

Hanaa Attou, Azidine Guezzaz, Said Benkirane, Mourade Azrour, and Yousef Farhaoui. Cloud-based intrusion detection approach using machine learning techniques. *Big Data Mining and Analytics*, 6(3):311–320, 2023a. doi: 10.26599/BDMA.2022.9020038.

Hanaa Attou, Azidine Guezzaz, Said Benkirane, Mourade Azrour, and Yousef Farhaoui. Cloud-based intrusion detection approach using machine learning techniques. *Big Data Mining and Analytics*, 6(3):311–320, 2023b. doi: 10.26599/BDMA.2022.9020038.

Shahid Allah Bakhsh, Muhammad Almas Khan, Fawad Ahmed, Mohammed S. Alshehri, Hisham Ali, and Jawad Ahmad. Enhancing iot network security through deep learning-powered intrusion detection system. *Internet of Things*, 24:100936, 2023. ISSN 2542-6605. doi: https://doi.org/10.1016/j.iot.2023.100936. URL https://www.sciencedirect.com/science/article/pii/S2542660523002597.

Chiradeep BasuMallick. Cloud access security broker: Pillars, architecture, uses, 2022. URL https://www.spiceworks.com/it-security/cloud-security/articles/what-is-casb/.

Mahesh Babu Bondada and S. Mary Saira Bhanu. Analyzing user behavior using keystroke dynamics to protect cloud from malicious insiders. In *2014 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 1–8, 2014. doi: 10.1109/CCEM.2014.7015481.

S. & Lawson C. Riley. Magic quadrant for cloud access security brokers. *Gartner*, 2020.

Zhuo Chen, Fu Jiang, Yijun Cheng, Xin Gu, Weirong Liu, and Jun Peng. Xgboost classifier for ddos attack detection and analysis in sdn-based cloud. In *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 251–256, 2018. doi: 10.1109/BigComp.2018.00044.

Chien-Yi Chiu, Chi-Tien Yeh, and Yuh-Jye Lee. Frequent pattern based user behavior anomaly detection for cloud system. In *2013 Conference on Technologies and Applications of Artificial Intelligence*, pages 61–66, 2013. doi: 10.1109/TAAI.2013.25.

Ashley Chonka, Yang Xiang, Wanlei Zhou, and Alessio Bonti. Cloud security defence to protect cloud computing against http-dos and xml-dos attacks. *Journal of Network and Computer Applications*, 34(4):1097–1107, 2011. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2010.06.004. URL `https://www.sciencedirect.com/science/article/pii/S1084804510001025`. Advanced Topics in Cloud Computing.

K L Dempsey, Nirali Shah Chawla, L A Johnson, Ronald Johnston, Alicia Clay Jones, A D Orebaugh, M A Scholl, and K M Stine. Information security continuous monitoring (iscm) for federal information systems and organizations, 2011.

Docker. What is a container?, 2020.

Mohamed Amine Ferrag, 2022. URL `https://www.kaggle.com/datasets/mohamedamineferrag/edgeiiotset-cyber-security-dataset-of-iot-iiot`.

Zecheng He, Tianwei Zhang, and Ruby B. Lee. Machine learning based ddos attack detection from source side in cloud. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 114–120, 2017. doi: 10.1109/CSCloud.2017.58.

Amazon.com inc. Shared responsibility model - amazon web services (aws), 2018.

Ponemon Institute. Cost of a data breach report 2021. *IBM Security*, 2022.

Deepak Kumar Jain, Weiping Ding, and Ketan Kotecha. Training fuzzy deep neural network with honey badger algorithm for intrusion detection in cloud environment, 2023. ISSN 1868-808X.

P. Kanimozhi and T. Aruldoss Albert Victoire. Oppositional tunicate fuzzy c-means algorithm and logistic regression for intrusion detection on cloud. *Concurrency and Computation: Practice and Experience*, 34(4):e6624, 2022. doi: https://doi.org/10.1002/cpe.6624. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6624`.

Taehyun Kim, Yeongrak Choi, Seunghee Han, Jae Yoon Chung, Jonghwan Hyun, Jian Li, and James Won-Ki Hong. Monitoring and detecting abnormal behavior in mobile cloud infrastructure. In *2012 IEEE Network Operations and Management Symposium*, pages 1303–1310, 2012. doi: 10.1109/NOMS.2012.6212067.

Jeffrey C Kimmell, Mahmoud Abdelsalam, and Maanak Gupta. Analyzing machine learning approaches for online malware detection in cloud. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 189–196, 2021. doi: 10.1109/SMARTCOMP52413.2021.00046.

Anil Kumar, Rajabov Sherzod Umurzoqovich, Nguyen Duc Duong, Pratik Kanani, Arulmani Kuppusamy, M. Praneesh, and Minh Ngyen Hieu. An intrusion identification and prevention for cloud computing: From the perspective of deep learning. *Optik*, 270:170044, 2022. ISSN 0030-4026. doi: https://doi.org/10.1016/j.ijleo.2022.170044. URL `https://www.sciencedirect.com/science/article/pii/S003040262201302X`.

Hongyu Liu and Bo Lang. Machine learning and deep learning methods for intrusion detection systems: A survey, 2019. ISSN 20763417.

M. Mayuranathan, S.K. Saravanan, B. Muthusenthil, and A. Samydurai. An efficient optimal security system for intrusion detection in cloud computing environment using hybrid deep learning technique. *Advances in Engineering Software*, 173:103236, 2022. ISSN 0965-9978. doi: https://doi.org/10.1016/j.advengsoft.2022.103236. URL `https://www.sciencedirect.com/science/article/pii/S0965997822001405`.

Peter Mell and Tim Grance. The nist definition of cloud computing, 2011.

Preeti Mishra, Emmanuel S. Pilli, Vijay Varadharajan, and Udaya Tupakula. Efficient approaches for intrusion detection in cloud environment. In *2016 International Conference on Computing, Communication and Automation (ICCCA)*, pages 1211–1216, 2016. doi: 10.1109/CCAA.2016.7813926.

Steve Morgan. Cybercrime To Cost The World $10.5 Trillion Annually By 2025, 2020. URL `"https://cybersecurityventures.com/cybercrime-damage-costs-10-trillion-by-2025/"`.

Monoj Muchahari and Smriti Sinha. A survey on web services and trust in cloud computing environment, 2013.

Ali Bou Nassif, Manar Abu Talib, Qassim Nasir, Halah Albadani, and Fatima Mohamad Dakalbab. Machine learning for cloud security: A systematic review. *IEEE Access*, 9:20717–20735, 2021. doi: 10.1109/ACCESS.2021.3054129.

National Institute of Standards and Technology. Zero trust architecture - nist special publication 800-207. *NIST*, 2020.

Ogobuchi Daniel Okey, Dick Carrillo Melgarejo, Muhammad Saadi, Renata Lopes Rosa, João Henrique Kleinschmidt, and Demóstenes Zegarra Rodríguez. Transfer learning approach to ids on cloud iot devices using optimized cnn. *IEEE Access*, 11:1023–1038, 2023a. doi: 10.1109/ACCESS.2022.3233775.

Ogobuchi Daniel Okey, Dick Carrillo Melgarejo, Muhammad Saadi, Renata Lopes Rosa, João Henrique Kleinschmidt, and Demóstenes Zegarra Rodríguez. Transfer learning approach to ids on cloud iot devices using optimized cnn. *IEEE Access*, 11:1023–1038, 2023b. doi: 10.1109/ACCESS.2022.3233775.

Abhishek Bapu Ove, 2021. URL https://www.youtube.com/watch?v=fGK3I8Nw0to.

N Pandeeswari and Ganesh Kumar. Anomaly detection system in cloud environment using fuzzy clustering based ann, 2016. ISSN 1572-8153.

EUropean PARLIAMENT. Regulation (eu) 2016/679 of the european parliament and of the council. *official Journal of the European Union*, 119, 2016. ISSN 1664848X.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Martin Roesch. Snort - lightweight intrusion detection for networks, 1999.

Scott Rose, Oliver Borchert, Stuart Mitchell, and Sean Connelly. Zero trust architecture, 2020. URL https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=930420.

Nate Rosidi, 2023. URL https://www.kdnuggets.com/2023/06/advanced-feature-selection-techniques-machine-learning-models.html.

Tara Salman, Deval Bhamare, Aiman Erbad, Raj Jain, and Mohammed Samaka. Machine learning for anomaly detection and categorization in multi-cloud environments. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 97–103, 2017. doi: 10.1109/CSCloud.2017.15.

K. Samunnisa, G. Sunil Vijaya Kumar, and K. Madhavi. Intrusion detection system in distributed cloud computing: Hybrid clustering and classification methods. *Measurement: Sensors*, 25:100612, 2023. ISSN 2665-9174. doi: https://doi.org/10.1016/j.measen.2022.100612. URL https://www.sciencedirect.com/science/article/pii/S266591742200246X.

M Subrahmanya Sarma, Y Srinivas, M Abhiram, Lakshminarayana Ullala, M. Sahithi Prasanthi, and J Rojee Rao. Insider threat detection with face recognition and knn user classification. In *2017 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 39–44, 2017. doi: 10.1109/CCEM.2017.16.

Da Seol, Jeong Choi, Chan Kim, and Sang Hong. Alleviating class-imbalance data of semiconductor equipment anomaly detection study. *Electronics*, 12, 2023. doi: 10.3390/electronics12030585.

Amazon Web Services. Aws lambda – serverless compute - amazon web services, 2021.

Pourya Shamsolmoali and Masoumeh Zareapoor. Statistical-based filtering system against ddos attacks in cloud computing. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1234–1239, 2014. doi: 10.1109/ICACCI.2014.6968282.

Taeshik Shon and Jongsub Moon. A hybrid machine learning approach to network anomaly detection. *Information Sciences*, 177(18):3799–3821, 2007. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2007.03.025. URL https://www.sciencedirect.com/science/article/pii/S0020025507001648.

Noor Suhana Sulaiman, Akhyari Nasir, Wan Othman, Syahrul Fahmy, Nur Aziz, Azliza Yacob, and Norfarina Samsudin. Intrusion detection system techniques : A review. *Journal of Physics: Conference Series*, 1874:012042, 05 2021. doi: 10.1088/1742-6596/1874/1/012042.

Md. Alamin Talukder, Md. Manowarul Islam, Md Ashraf Uddin, Khondokar Fida Hasan, Selina Sharmin, Salem A. Alyami, and Mohammad Ali Moni. Machine learning-based network intrusion detection for big and imbalanced data using oversampling, stacking feature embedding and feature extraction, 2024. ISSN 2196-1115. URL https://doi.org/10.1186/s40537-024-00886-w.

Ankit Thakkar and Ritika Lohiya. A review on challenges and future research directions for machine learning-based intrusion detection system, 2023. ISSN 1886-1784. URL https://doi.org/10.1007/s11831-023-09943-8.

D Anurag Varma, Ravi Ashish, and V Sandeep. Detection of ddos attacks using machine learning techniques: A hybrid approach, 2022.

Somayeh Youssefi. Permutation importance vs impurity-based feature importance, Sep 2023. URL https://medium.com/@syoussefi600/permutation-importance-vs-impurity-based-feature-importance-1c1a8d027479.

Appendices

Appendix A

Paper IDs

Table 32: Papers corresponding to each ID

ID	Paper	Type	Year
P1	"Anomaly Detection System in Cloud Environment Using Fuzzy Clustering Based ANN"	Journal	2015
P2	"Machine Learning for Anomaly Detection and Categorization in Multi-Cloud Environments"	Conference	2017
P3	"A hybrid machine learning approach to network anomaly detection"	Journal	2007
P4	"Frequent Pattern Based User Behavior Anomaly Detectopm for Cloud System"	Conference	2014
P5	"Monitoring and detecting abnormal behavior in mobile Cloud infrastructure"	Conference	2012
P6	"Insider Threat Detection with Face Recognition and KNN User Classification"	Conference	2018
P7	"ML Confidential: Machine Learning on Encrypted Data"	Conference	2013
P8	Trust Issues that Create Threats for Cyber Attacks in Cloud Computing"	Conference	2011
P9	"Evaluation of machine learning classifiers for mobile malware detection"	Journal	2014
P10	"Applying machine learning classifiers to dynamic Android malware detection at scale"	Conference	2013
P11	"Combining file content and file relations for Cloud based malware detection"	Conference	2011
P12	"Statistical-based filtering system against DDOS attacks in CLoud computing"	Conference	2014
P13	"NvCloudIDS: A security architecture to detect intrusions at network and virtualization layer in Cloud environment"	Conference	2016
P14	"Continuous security assessment of Cloud based applications using distributed hashing algorithm in SDLC"	Journal	2017
P15	"Machine Learning Based DDoS Attack Detection from Source Side in Cloud"	Conference	2017
P16	"Secure Data Mining in Cloud Using Homomorphic Encryption"	Conference	2015
P17	"Cache-Based Application Detection in the Cloud Using Machine Learning"	Conference	2017
P18	"Privacy-preserving Machine Learning in Cloud"	Journal	2017
P19	"Improvement of security in Cloud systems based on steganography"	Conference	2014
P20	"VLOC: An Approach to Verify the Physical Location of a Virtual Machine in Cloud"	Conference	2015
P21	"Computing encrypted Cloud data efficiently under multiple keys"	Conference	2013
P22	"Privacy Preserving Back-Propagation Neural Network Learning Made Practical with Cloud Computing"	Journal	2013
P23	"Cloud-Based Cyber-Physical Intrusion Detection for Vehicles Using Deep Learning"	Journal	2017
P24	"Detecting Denial of Service Attacks in the Cloud"	Conference	2016
P25	"Security-Aware Information Classifications Using Supervised Learning for Cloud-Based Cyber Risk Management in Financial Big Data"	Conference	2016
P26	"Self-learning method for DDoS detection model in Cloud computing"	Conference	2017
P27	"Analyzing User Behavior Using Keystroke Dynamics to Protect Cloud from Malicious Insiders"	Conference	2015
P28	"EXpectation Propagation LOgistic REgRession (EXPLORER): Distributed privacy-preserving online model learning"	Journal	2013
P29	"Privacy Preserving Deep Computation Model on Cloud for Big Data Feature Learning"	Journal	2015
P30	"PDLM: Privacy-Preserving Deep Learning Model on Cloud with Multiple Keys"	Journal	2018
P31	"MSCryptoNet: Multi-Scheme Privacy-Preserving Deep Learning in Cloud Computing"	Journal	2019
P32	"Cloud security defence to protect Cloud computing against HTTP-DoS and XML-DoS attacks"	Journal	2011
P33	"Malware Detection in Cloud Computing Infrastructures"	Journal	2015
P34	"Secure Collaborative Outsourced Data Mining with Multi-owner in Cloud Computing"	Conference	2012
P35	"XGBoost Classifier for DDoS Attack Detection and Analysis in SDN-Based Cloud"	Conference	2018
P36	"Incremental k-NN SVM method in intrusion detection"	Conference	2018
P37	"Security Enhancement in Healthcare Cloud using Machine Learning"	Conference	2018
P38	"Malicious Executables Classification Based on Behavioral Factor Analysis"	Conference	2010
P39	"A comparative analysis of SVM and its stacking with other classification algorithm for intrusion detection"	Conference	2016
P40	"Malware behavioural detection and vaccine development by using a support vector model classifier"	Journal	2015
P41	"Designing encryption and IDS for Cloud security"	Conference	2017
P42	"A novel intrusion severity analysis approach for Clouds"	Journal	2013
P43	"A novel framework for intrusion detection in Cloud"	Conference	20012
P44	A neural network based distributed intrusion detection system on Cloud platform"	Conference	2013
P45	"Cloud-based mobile system for biometrics authentication"	Conference	2013
P46	"Secure and controllable KNN query over encrypted Cloud data with key confidentiality"	Journal	2016
P47	"An Efficient DDoS TCP Flood Attack Detection and Prevention System in a Cloud Environment"	Journal	2017
P48	"Efficient approaches for intrusion detection in Cloud environment"	Conference	2017
P49	"Secure Naive Bayesian Classification over Encrypted Data in Cloud"	Conference	2016
P50	"DDoS Attacks Detection in Cloud Computing Using Data Mining Techniques"	Conference	2016
P51	"Detecting DDoS attacks against data center with correlation analysis"	Journal	2015
P52	"Detection of known and unknown DDoS attacks using Artificial Neural Networks"	Journal	2016
P53	"A Combined Decision for Secure Cloud Computing Based on Machine Learning and Past Information"	Conference	2019
P54	"Analysis and Detection of DDoS Attacks on Cloud Computing Environment using Machine Learning Techniques"	Conference	2019
P55	"Machine Learning-Based EDoS Attack Detection Technique Using Execution Trace Analysis"	Journal	2019
P56	"Design of network threat detection and classification based on machine learning on cloud computing"	Journal	2018
P57	"VMAnalyzer: Malware Semantic Analysis using Integrated CNN and Bi-Directional LSTM for Detecting VM-level Attacks in Cloud"	Conference	2019

ID	Paper	Type	Year
P58	"Detecting cyber-physical attacks in CyberManufacturing systems with machine learning methods"	Journal	2019
P59	DeMETER in clouds: detection of malicious external thread execution in runtime with machine learning in Paas clouds"	Journal	2019
P60	"A deep learning approach for proactive multi-cloud cooperative intrusion detection system"	Journal	2019
P61	"K-NN classifier for data confidentiality in cloud computing"	Conference	2014
P62	"A machine learning algorithm TSF k-Nn based on automated data classification for securing mobile cloud computing model"	Conference	2019
P63	"Cloud based emails boundaries and vulnerabilities"	Conference	2013
P64	"Analyzing Machine Learning Approaches for Online Malware Detection in Cloud"	Conference	2021
P65	"Transfer Learning Approach to IDS on Cloud IoT Devices Using Optimized CNN"	Journal	2023
P66	"Detection of DDOS attacks using machine learning techniques: A hybrid approach"	Conference	2022
P67	"Cloud-Based Intrusion Detection Approach Using Machine Learning Techniques"	Journal	2023
P68	"An intrusion identification and prevention for cloud computing: From the perspective of deep learning"	Journal	2022
P69	"Oppositional tunicate fuzzy C-means algorithm and logistic regression for intrusion detection on cloud"	Journal	2021
P70	"Intrusion detection system in distributed cloud computing: Hybrid clustering and classification methods"	Journal	2023
P71	"An efficient optimal security system for intrusion detection in cloud computing environment using hybrid deep learning technique"	Journal	2022
P72	"Training fuzzy deep neural network with honey badger algorithm for intrusion detection in cloud environment"	Journal	2023
P73	"A Cloud Intrusion Detection Systems Based on DNN Using Backpropagation and PSO on the CSE-CIC-IDS2018 Dataset"	Journal	2023