Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Deep Learning Approach for Network Attack Prediction

Bernardo Rodrigues Vitorino

Orientador(es) | Pedro Patinho

Pedro Salgueiro

Évora 2025

Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Deep Learning Approach for Network Attack Prediction

Bernardo Rodrigues Vitorino

Orientador(es) | Pedro Patinho

Pedro Salgueiro

Évora 2025

A dissertação foi objeto de apreciação e discussão pública pelo seguinte júri nomeado pelo Diretor da Escola de Ciências e Tecnologia:

Presidente	Teresa Gonçalves (Universidade de Évora)
Vogais	Luís Rato (Universidade de Évora) (Arguente)
	Pedro Patinho (Universidade de Évora) (Orientador)

Évora 2025

Dedicated to my parents, whose support and encouragement have been the foundation of all my achievements.

Acknowledgments

I express my gratitude to my supervisors, **Profesor Pedro Patinho** and **Professor Pedro Salgueiro**, for their guidance, encouragement, and insightful feedback throughout the development of this dissertation. Their experience and dedication were fundamental to the completion of this work.

I am also grateful to the **University of Évora**, particularly the **Department of Informatics** and the **School of Science and Technology**, for providing the resources and environment that made this research possible.

A great thank you to my work colleagues, **David Cravinho** and **Fernando Oliveira**, for inspiring my curiosity and deepening my passion for cybersecurity.

Special thanks to my family and friends for their constant support and encouraging words throughout this journey.

Finally, I would like to acknowledge all those who contributed, directly or indirectly, to the realization of this work.

Contents

List of Figures

List of Tables

Acronyms

DDoS	Distributed Denial of Service
Tbps	Terabits per second
IoT	Internet of Things
IDPS	Intrusion Detection and Prevention Systems
IDS	Intrusion Detection Systems
NIDS	Network Intrusion Detection Systems
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
DoS	Denial of Service
LSTM	Long Short-Term Memory
KNN	K-Nearest Neighbors
SVM	Support Vector Machines
NB	Naive Bayes
MLP	Multi-Layer Perceptron
MLPs	Multi-Layer Perceptrons
DT	Decision Trees
CNNs	Convolutional Neural Networks
RNNs	Recurrent Neural Networks
GRU	Gated Recurrent Units
DBNs	Deep Belief Networks
CSV	Comma-Separated Values
SMOTE	Synthetic Minority Over-sampling Technique
PCAP	Packet Capture
ANNs	Artificial Neural Networks
NN	Neural Network
DNNs	Deep Neural Networks
AEs	Autoencoders
DR	Detection Rate

Abstract

In today's hyperconnected world, the rapid expansion of networked technologies and data interchange has significantly expanded the attack surface for malicious entities. The growing frequency and complexity of cyberattacks highlight the necessity for intelligent and adaptive protection systems that can anticipate and counteract threats in real time. This dissertation examines the design and implementation of a Network Intrusion Prediction System utilizing deep learning techniques, specifically emphasizing Long Short-Term Memory (LSTM) networks. The proposed method uses the CIC-IDS2017 dataset, a recognized standard for intrusion detection, and integrates an extensive data preprocessing pipeline to organize traffic flows for sequential pattern recognition.

The LSTM-based model, through careful training and evaluation, achieves a test accuracy of **99.64%**, exhibiting exceptional performance in multiclass attack prediction while ensuring robustness across various attack types. The dissertation introduces a modular proof-of-concept system architecture that incorporates data gathering, flow analysis, deep learning inference, and near real-time alert generation, in addition to the model itself. The findings highlight both the efficacy of LSTM networks in predicting sophisticated attack patterns and the practical viability of using these models in operational network environments.

The presented work offers theoretical and practical insights into the application of deep learning in network security, illustrating that data-driven intrusion prediction systems can improve early threat detection, reduce reliance on static rule-based mechanisms, and improve the resilience of critical infrastructures against evolving cyber threats.

Keywords: Cybersecurity, Deep Learning, Network Attack Prediction, LSTM, Intrusion Detection System

Sumário

Abordagem de Deep Learning para Predição de Ataques em Redes

No mundo hiperconectado de hoje, a rápida expansão das tecnologias em rede e da troca de dados aumentou significativamente a superfície de ataque disponível para entidades maliciosas. A crescente frequência e complexidade dos ciberataques evidenciam a necessidade de sistemas de proteção inteligentes e adaptativos, capazes de antecipar e contrariar ameaças em tempo real. Esta dissertação analisa o desenho e a implementação de um Sistema de Previsão de Intrusões em Redes recorrendo a técnicas de aprendizagem profunda, com especial ênfase nas redes LSTM. O método proposto utiliza o conjunto de dados CIC-IDS2017, um padrão reconhecido para deteção de intrusões, e integra uma extensa etapa de pré-processamento de dados para organizar os fluxos de tráfego de forma a permitir o reconhecimento de padrões sequenciais.

O modelo baseado em LSTM, através de um treino e avaliação rigorosos, alcança uma precisão de teste de **99.64%**, demonstrando um desempenho excecional na previsão de ataques multiclasse, assegurando simultaneamente robustez face a diferentes tipos de ameaças. A dissertação apresenta ainda uma arquitetura modular de prova de conceito que incorpora recolha de dados, análise de fluxos, inferência por aprendizagem profunda e geração de alertas em quase tempo real, para além do próprio modelo. Os resultados obtidos evidenciam tanto a eficácia das redes LSTM na previsão de padrões de ataque sofisticados, como a viabilidade prática da utilização destes modelos em ambientes operacionais de rede.

O trabalho apresentado oferece contributos teóricos e práticos para a aplicação da aprendizagem profunda na segurança de redes, ilustrando que sistemas de previsão de intrusões baseados em dados podem melhorar a deteção precoce de ameaças, reduzir a dependência de mecanismos de deteção baseados em regras estáticas e aumentar a resiliência de infraestruturas críticas perante ameaças cibernéticas em constante evolução.

Palavras chave: Cibersegurança, Aprendizagem Profunda, Predição de Ataques em Redes, LSTM, Sistema de Deteção de Intrusões

Chapter 1

Introduction

1.1 Background and Motivation

Cyberattacks have become a persistent threat to educational institutions, healthcare facilities, businesses, and even sovereign states. The increase in cybercrime, coupled with the swift advancement of methods used by malicious entities, has highlighted the essential requirement for strong network security to safeguard sensitive information and crucial infrastructure. Network-based attacks, while constituting only part of the wider cyber threat landscape, are very disruptive. This includes several occurrences, including data breaches and Distributed Denial of Service (DDoS) attacks, which can interrupt systems, compromise confidential information, and lead to substantial financial losses.

In 2025, more than one third of the UK's schools experienced significant cyber-attacks, many associated with Russian-speaking ransomware organizations. These assaults interrupted education and jeopardized confidential information, with ransom requests averaging £5.1 million. The recovery frequently faces expenses close to £3 million [1].

Fortunately, not all cyberattacks achieve their intended objectives. An impressive instance of an effective defense occurred in October 2024, when Cloudflare[1] announced the blockage of the most significant DDoS attack ever documented, peaking at an extraordinary 5.6 Terabits per second (Tbps). The target was an internet service provider in Eastern Asia, executed via a variation of the Mirai botnet, utilizing over 13,000 compromised Internet of Things (IoT) devices. Despite the unparalleled magnitude and complexity of the attack, Cloudflare's automated defensive mechanisms successfully identified and neutralized the threat in real time, eliminating the need for human intervention [4].

[1]Cloudflare is a U.S.-based web infrastructure and website security company that provides services such as content delivery network (CDN), DDoS mitigation, Internet security, and distributed domain name server (DNS) services. It acts as a reverse proxy between a website's visitors and the hosting server, improving performance and protecting against malicious traffic [2; 3].

1.2 Problem Statement

These instances highlight the necessity of automated systems competent in analyzing and identifying such threats. A commonly employed technique for protecting against intrusions is rule-based Intrusion Detection and Prevention Systems (IDPS). These systems depend on established signatures and patterns to identify malicious activity and have demonstrated efficacy against known or less complex threats. Nonetheless, the complexity of contemporary techniques has made traditional, static Intrusion Detection Systems (IDS) and prevention mechanisms, though somewhat effective, ineffective in adapting to the dynamic and complex nature of modern attacks, due to their incapacity of combining predefined rules with sophisticated and evasive threats [5].

The increasing volume and variety of network traffic, along with the growing sophistication of attackers, calls for more advanced and adaptive solutions for the effective detection and prediction of attacks, giving cybersecurity professionals the opportunity to respond or enabling automated systems to neutralize threats before they inflict damage.

1.3 Research Motivation and Approach

In light of these new concerns, Artificial Intelligence (AI) has become a formidable ally in cybersecurity. Artificial Intelligence denotes systems that replicate human intellectual functions, while Machine Learning (ML) facilitates the enhancement of these systems' performance over time via experience and data. Deep Learning (DL), a category of machine learning techniques utilizing artificial neural networks with multiple layers, has attracted interest for its capacity to identify intricate patterns in data with limited human oversight. In contrast to conventional machine learning models, such as decision trees or support vector machines that require human feature extraction, deep learning models can immediately learn from unprocessed network traffic data, detecting nuanced and previously unrecognized symptoms of compromise [6].

The efficacy of AI or DL-based systems is significantly dependent on the quality and amount of the training data. Well-labeled and representative datasets allow models to accurately distinguish between normal and malicious behavior, while poor-quality data can lead to high false positive or false negative rates. Thus, rigorous dataset selection and preparation are essential elements in developing an efficient predictive system.

To achieve this objective, the CIC-IDS2017 [7] dataset, an established resource in the field of IDS research, has been selected for experimentation and analysis. This dataset, produced by the Canadian Institute for Cybersecurity, accurately depicts genuine network traffic, covering both hostile and benign activities. This approach is particularly effective for evaluating modern intrusion detection methods, as it includes a diverse range of attack types, such as Denial of Service (DoS), DDoS, brute force, port scanning, and web-based attacks.

1.4 Objectives and Contributions

This dissertation aims to improve the field of network intrusion prediction by going beyond theoretical assessment and illustrating its practicality. In particular, it creates and carefully tests a deep learning model based on Long Short-Term Memory (LSTM) networks that is trained on the CIC-IDS2017 dataset and is then used in an operational proof-of-concept system. The work not only shows that LSTM architectures are capable of accurately predicting different and imbalanced attack types, but also provides a deployable and modular system architecture that can predict intrusions in almost real time in real-world network environments.

All the source code, scripts, and configuration files developed during this dissertation are publicly available in the following repository for reproducibility and future research reference:

`https://github.com/Shogvn404/Cerberus-DeepShield`

1.5 Research Questions

To guide this research, the following questions are posed:

- Can an LSTM-based deep learning model, chosen for its success in similar intrusion detection studies, effectively predict multiple types of network attacks on the CIC-IDS2017 dataset compared to other possible architectures?
- How well can such a model generalize to rare or low-frequency attack types without being dominated by majority classes?
- Is it feasible to design and deploy a modular, real-time proof-of-concept system architecture that operationalizes the trained model in a realistic network environment?
- What are the practical strengths and limitations of deploying deep learning models for intrusion prediction in real-world conditions?

1.6 Research Hypothesis

Based on this, the central research hypothesis is as follows:

> A deep learning model based on a Long Short-Term Memory (LSTM) network, when trained on a representative dataset such as CIC-IDS2017 and deployed within a real-world infrastructure, can achieve high multiclass network attack prediction accuracy with low false positive rates. Moreover, a practical, scalable, and modular proof-of-concept system architecture can be designed and implemented to operationalize this model for real-time threat detection in real network environments.

This hypothesis forms the foundation for the practical and experimental contributions detailed in the following chapters.

1.7 Structure of the Dissertation

The remainder of this dissertation is organized as follows. Chapter 2 introduces the theoretical foundations of cybersecurity and artificial intelligence. Chapter 3 reviews the state of the art, including commonly used datasets and related work. Chapter 4 details the data engineering process and the development of the deep learning model. Chapter 5 presents the design and implementation of the proposed system architecture. Chapter 6 describes the evaluation methodology, experimental setup, and results. Finally, Chapter 7 concludes the dissertation by summarizing the main contributions and outlining directions for future work.

1.8 Summary

The growing complexity of cyber threats and the lack of effectiveness of traditional detection methods show how important it is to use more flexible, data-driven methods in network security. Deep learning methodologies, notably Long Short-Term Memory (LSTM) networks, present a viable approach for modeling the temporal dynamics of network traffic and predicting malicious activities prior to inflicting substantial harm. This dissertation addresses these difficulties by developing and assessing a deep learning-based intrusion prediction system, based on the CIC-IDS2017 dataset and intended to operate in realistic network contexts. The work provided here not only aims to confirm the efficacy of LSTM models for multiclass attack prediction but also promotes a modular system architecture designed to illustrate the practical viability of implementing such models for real-time cybersecurity defense.

Chapter 2

Theoretical Foundation

The use of deep learning in network attack prediction requires a solid understanding of foundational concepts, both in network security and artificial intelligence. This section lays the fundamental knowledge for understanding how deep learning can be effectively applied in the prediction and mitigation of network attacks. It begins by providing a theoretical foundation about cybersecurity and the related terms later used in this paper, followed by a brief explanation of artificial intelligence as well as some related terms. Finally the tools used in the Chapters 4 and 5 will be introduced in this Chapter, followed by a conclusion Section.

2.1 Cybersecurity Theoretical Foundation

Cybersecurity is a concept that implements practices, procedures, and technical mechanisms to protect, detect, mitigate, and defend against damage, usage, or unauthorized modification of information and communication systems. It's a field that is becoming more complex day by day due to the exponential growth of the number of interlinked devices, systems, and networks. All of this has a direct relation with the economy and digital infrastructure growth, which leads to more and more complex cyberattacks [8]. Information systems security covers many activities and is often described using structures like the NIST Cybersecurity Framework [9] that defines five main functions:

- **Identify**: This function provides the foundation for the other cybersecurity function, identifying the functions and risks associated with systems, individuals, assets, and data. This includes activities like asset management, business environment, governance, and risk assessment.
- **Protect**: This function helps the planning and implementation of appropriate controls to limit or contain the impact of a potential cybersecurity event. This includes technical and procedural controls to proactively protect against internal and external cybernetic threats. The categories that are included

in this function are identity management, authentication and access control, awareness and training, data security, and information protection processes and procedures.

- **Detect**: The detect function allows the timely discovery of cybersecurity events, developing and implementing appropriate activities to identify their occurrence. Includes activities for the timely detection of anomalies and intrusions, impact evaluation, continuous security monitoring to verify the effectiveness of protective measures, and appropriate maintenance of detection processes to ensure the awareness of security events.
- **Respond**: This function creates a road map for managing and limiting the impact of a potential cybersecurity event. This includes the timely planning to develop effective processes capable of problem resolution, incident analysis for cause determination, scope and impact, incident contention, and communication coordination during and after the attack.
- **Recover**: This function supports the development and implementation of activities to restore services and operations affected by a cybersecurity event to a normal operational state. This involves implementing recovery plans, conducting recovery activities, and making long-term improvements.

Advancements in artificial intelligence (AI) have resulted in a more frequent utilization of AI-driven solutions in cybersecurity, aiding security teams in effectively mitigating threats and improving security measures. AI can automate repetitive operations, enhance threat detection and response, and increase the precision of actions, thereby reinforcing security measures against diverse security problems and cyberattacks.

Cybersecurity is a vast field in the technology world, and to better understand the context of the present dissertation, some important concepts are presented here.

Cybercrime

Cybercrime refers to illegal activities that involve or target computers, computer networks, or connected devices. While many cybercrimes are financially motivated and carried out by hackers or cybercriminals seeking profit, some attacks are driven by political, personal, or ideological reasons, aiming to cause disruption or harm. These crimes can be committed by individuals or organized groups, ranging from highly skilled and well-equipped professionals to amateur hackers with limited expertise [10].

Cyberattack

A cyberattack is an intentional effort to obtain unauthorized access to a computer system, network, or digital device with the objective of stealing, revealing, changing, disabling, or destroying data, applications, or other digital assets [11].

Vulnerability

Vulnerabilities are bugs in software that malicious individuals can use to make a system perform in ways that aren't intended, including giving away information about its defenses. When MITRE finds a bug that can be applied as an exploit, it is registered as a CVE (Common Vulnerability or Exposure) and assigns it a CVSS score to indicate its potential risk [12].

Malicious Hacker

A hacker is someone who gets into computer systems without permission. Their reasons for doing so can range from planting malware and stealing or deleting data to stopping services from working. Fortunately, not all hacking is harmful, some hackers, called "Ethical hackers", try to find security holes so they can be fixed [13]. In the remainder of this dissertation, the term *hacker* will refer specifically to malicious actors.

Computer Network

A computer network is a group of machines that are connected to each other and share information and resources, such files, internet access, or devices [14].

Firewall

A firewall is a security tool that manages network traffic between a trusted internal network and an untrusted external network, such as the Internet. It stops unwanted access and protects against risks by filtering data according to a set of rules. It can be used as hardware, software, or cloud-based solution [15].

Intrusion Detection Systems

An Intrusion Detection System (IDS) is a network security tool designed to monitor traffic and devices for signs of malicious behavior, unusual activity, or breaches of security policies [16].

Intrusion Prevention Systems

An Intrusion Prevention System (IPS) analyzes network traffic for potential threats and takes immediate action to block them. It can alert security teams, terminate harmful connections, remove malicious content, or activate other security measures to prevent damage [17].

2.1.1 Common Types of Network Attacks

To facilitate a better understanding of the dataset used in this study, it is essential to first introduce several common types of network attacks. This section provides an overview of both the attack types represented in the dataset and a few others included for contextual and comparative purposes.

DoS and DDoS attacks

A denial-of-service (DoS) attack aims to overload a system's resources, preventing it from handling legitimate service requests. A distributed denial-of-service (DDoS) attack works similarly but involves numerous malware-infected devices controlled by the attacker to amplify the attack. These attacks render the target system unable to serve its users, effectively "denying the service".

In a DoS attack, the target is flooded with fake requests, consuming resources and often causing a total shutdown. Unlike other cyberattacks that seek to gain or increase unauthorized access, DoS and DDoS attacks are designed purely to disrupt a service. However, attackers may indirectly benefit, such as when hired by competitors to cause financial harm to the target [18].

MITM attacks

A man-in-the-middle (MITM) attack is a cybersecurity breach where an attacker secretly intercepts and potentially alters the communication between two parties. The attacker positions itself between the two entities, eavesdropping or manipulating the data without either side realizing it. During a MITM attack, both parties believe they are communicating normally, unaware that the attacker is interfering with their messages. To prevent such attacks, using strong encryption on access points or employing a virtual private network (VPN) can improve the communications security [19].

Session hijacking

Session hijacking is a type of MITM attack where an attacker takes control of an active session between a client and a server. By substituting their own IP address for the client's, the attacker tricks the server into continuing the session, believing it is still communicating with the legitimate client. This works because the server relies on the client's IP address for authentication and may not detect the intrusion during an ongoing session. To prevent session hijacking, it is good policy to use a VPN to encrypt communications with critical servers [20].

URL interpretation

URL interpretation, also known as URL poisoning, involves attackers modifying or guessing URLs to access personal or sensitive data. The term "URL interpretation" refers to attackers analyzing the structure and syntax of URLs to locate and exploit restricted areas, like admin sections or user accounts.

To execute such an attack, hackers may try URLs they suspect lead to sensitive areas, like an admin page, and then use default or weak credentials to gain access. Once inside, they can steal, manipulate, or delete data. To mitigate such attacks, it is essential to implement robust security measures such as multi-factor authentication (MFA) and enforce the use of strong, hard-to-guess passwords [21].

DNS spoofing

DNS spoofing involves hackers alternating DNS records to redirect users to a fake website. On the fraudulent site, victims might unknowingly provide sensitive information, which the attacker can exploit or sell. In some cases, hackers create low-quality or harmful sites to damage a competitor's reputation. This attack deceives victims into believing they are visiting a legitimate site, allowing the attacker to commit malicious acts under the guise of the targeted company. To prevent DNS spoofing, DNS servers must be regularly updated, as newer software versions often address known vulnerabilities that attackers try to exploit [22].

Brute force attacks

A brute-force attack involves an attacker attempting to guess login credentials through repeated trial and error. Often, bots are used to automate the process, trying numerous combinations of usernames and passwords until the correct one is found, granting the attacker access.

A common defense against brute-force attacks is the implementation of account lockdown policies, which temporarily disable access after several failed login attempts, preventing continuous guessing. In addition, creating strong and unpredictable passwords, ideally long combinations of random words rather than complex but predictable, greatly increases resistance to password cracking. Passwords that are easier for humans to remember can still be extremely difficult for attackers or automated tools to guess, provided they are sufficiently long and random [23].

Web attacks

Web attacks are malicious attempts to exploit vulnerabilities in web-based applications by manipulating user interactions and system requests. These attacks target the fundamental way web applications process user commands, such as financial transactions or login processes [24].

Common types of web attacks include:

- SQL injection
- Cross-site scripting (XSS)
- Cross-site request forgery (CSRF)
- Parameter tampering

SQL Injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. By inserting malicious SQL code into input fields, attackers can gain unauthorized access to data or modify database content [25].

Cross-Site Scripting is a vulnerability that enables attackers to inject malicious scripts (usually JavaScript) into trusted websites, allowing them to steal session cookies, deface pages, or redirect users to malicious sites [26].

In a CSRF attack, hackers trick users into performing actions that benefit the attacker, like changing login credentials. Parameter tampering involves modifying security parameters to bypass protective measures [27].

To prevent web attacks, organizations can:

- Regularly inspect web applications for vulnerabilities
- Implement anti-CSRF tokens to validate commands
- Use SameSite flags to restrict cross-site request processing

The key defense is understanding how attackers exploit the communication between user inputs and web application responses, and implementing robust security measures to prevent unauthorized access or manipulation.

Backdoor

A backdoor attack is a threat that allows unauthorized access to a computer system or network. Attackers create hidden vulnerabilities or opening in a system's code, which lets them bypass normal security measures. By inserting malicious code, they can gain control of the system and perform harmful activities like stealing data, installing additional malware, or launching further attacks [28].

Trojan Horse

A Trojan horse attack is a cybersecurity threat where malicious software in concealed within a program that appears legitimate and harmless. When the user unknowingly runs the seemingly innocent application, the hidden malware activates, creating a backdoor that allows hackers to infiltrate the computer or network. The term “Trojan horse” is derived from the ancient Greek mythology of soldiers hiding inside a large wooden horse, which was presented as a gift to the city of Troy. Once the horse was brought inside the city walls, the hidden soldiers emerged and attacked, reflecting how the malware disguises itself within an apparently harmless program. The attack works by tricking users into voluntarily introducing the malicious program into their system, exploiting trust and curiosity. Once inside, the Trojan can provide unauthorized access, potentially leading to data theft, system compromise, or other harmful actions [18].

Fuzzers

Fuzzing is a cybersecurity technique that involves sending unexpected or random input data to an application to discover vulnerabilities. While security researchers use fuzzing to proactively identify and fix software weaknesses, attackers can also exploit this method maliciously. In an attack scenario, cybercriminals use specialized tools called fuzzers to systematically probe an application with varied inputs. By overwhelming the software with unexpected data, they aim to identify potential security flaws that can be further analyzed and exploited. When a vulnerability is discovered through fuzzing, attackers can develop targeted strategies to compromise the system [28].

Reconnaissance

This attack, also known as information gathering or footprinting, is a cyberattack focused on gathering intelligence about a target system or network. Attackers collect information to understand the infrastructure, identify vulnerabilities, and discover potential entry points, all without causing direct damage to the system. The goal is to collect valuable insights that can be used for planning future, more invasive attacks [29].

Worms

A worm is a self-replicating malware that spreads through computer networks independently. Unlike other malicious programs, worms do not need user interaction to propagate. They can automatically replicate and spread across multiple computers and devices by exploiting security vulnerabilities in networked systems [30].

2.2 Artificial Intelligence Theoretical Foundation

AI provides computational methods for simulating intelligent behavior, among which ML and DL are particularly influential subfields [31]. In cybersecurity, these approaches enable the analysis of large-scale network data, the identification of complex patterns, and the prediction of malicious activities. This subsection introduces the essential principles of ML, artificial neural networks, and DL, along with the role of datasets and network architectures, establishing the theoretical basis for the predictive model developed in this dissertation.

Machine Learning

A branch of artificial intelligence that focuses on creating algorithms that can extract information from data and use that knowledge to make predictions or choices. Machine learning algorithms are trained on large datasets and use that training to identify patterns and relationships that can be used to make predictions on new data, as opposed to being explicitly programmed to perform a specific task [32][33].

In this type of studies machine learning procedures are utilized to learn the attackers past behavior, and then use that knowledge to anticipate the subsequent steps of an attack in real time [34].

Artificial Neural Networks

Computational models inspired by the biological nervous system, particularly the human brain, they consist of interconnected neurons linked by weighted connections, forming a complex, nonlinear structure. Artificial Neural Networks (ANNs) process data through learning and training methods, involving tasks like data collection, network design, and weight adjustments. These models are categorized into three types: static (e.g., multilayer perceptrons), dynamic (e.g., recurrent neural networks), and statistical (e.g., radial basis function networks). ANNs are widely used across various scientific fields for pattern classification, prediction, and optimization, and they can be combined with other techniques, such as adaptive neuro-fuzzy inference systems, for enhanced predictive capabilities [35].

Deep Learning

Deep Learning is a specific type of Machine Learning that utilizes ANNs with multiple layers. Unlike traditional ML algorithms, which may be limited in their ability to extract complex features from data, Deep Learning can learn hierarchical representations of data, enabling it to model intricate relationships and make more accurate predictions. Deep Learning techniques have made significant advancements across various domains, including network security, for instance in Network Intrusion

Detection Systems (NIDS). These techniques/approaches aim to observe, encode, predict, or classify patterns or sequences of patterns by learning effective feature representations [34].

Dataset

A dataset is a collection of data that can be organized in tables, arrays or more specific formats, such as CSV or JSON, for an easier retrieval and analysis. Datasets are one of the main pillars of data analysis, machine learning, artificial intelligence and other applications that require reliable, accessible data [36].

2.2.1 Deep Learning Networks Architectures

Deep Learning is in itself a extensive field, where previous research and work has proved that the simplest change in the design of a neural network can change the outcome. In this Section are described some Neural Network (NN) architectures that were used in previous work to build a necessary background for the related work analysis. Here I describe architectures like Deep Neural Networks (DNNs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Deep Belief Networks (DBNs), Long Short-Term Memory (LSTMs), Autoencoders (AEs) and Multi-Layer Perceptrons (MLPs).

Deep Neural Networks (DNNs)

Deep neural networks are extensions of artificial neural networks (ANNs) with a more complex hidden layer structure. Unlike traditional ANNs, which typically consist of an input layer, one or two hidden layers, and an output layer, DNNs have significantly more layers providing greater depth and requiring more computational power [37].

To handle complex tasks like image processing, computer vision, and natural language processing, DNNs utilize a variety of specialized layers, including convolutional layers, max-pooling layers, and dense layers. These additional layers allow the model to understand problems in greater detail, enabling optimal solutions to intricate challenges. Each layer adds complexity, helping process inputs more effectively to generate accurate outputs.

DNNs have gained immense traction due to their high efficiency in solving diverse deep learning tasks, proving indispensable for projects that require advanced data processing and analysis.

Convolutional Neural Networks (CNNs)

CNNs are a deep learning algorithm that is mostly specialized in tasks that involve object recognition, like image classification, detection and segmentation [38]. Convolutional layers give to this architecture the ability to detect patterns regardless of the position, orientation, scale or translation making it particularly good at recognizing objects or features within images in different conditions.

Nevertheless, this type of NN has also proved its capability in other fields such as Natural Language Processing (NLP), Time Series Analysis and Speech Recognition. In the spectrum of this dissertation, it's important to highlight its capability of analyzing structured temporal data, identifying patterns in sequential inputs.

They are highly effective at identifying complex patterns, regardless of whether they range from short durations or extend over longer periods, making it very adaptable to various fields. This feature is particularly helpful in network intrusion detection, where it is essential to analyze sequential data, such as packet flows or traffic logs. This architecture is a powerful tool for proactive network protection because they may identify tiny deviations that may be signs of possible security breaches or cyberattacks by gradually learning and identifying patterns of normal and aberrant activity [39].

Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a type of neural network designed to process sequential data by incorporating a feedback mechanism. In RNNs, the output from one step is looped back as input for the next, enabling the network to retain and utilize information from previous steps. This design allows RNNs to excel at tasks requiring context from earlier elements in a sequence, such as predicting the next word in a sentence or analyzing time-series data [40].

The key feature of RNNs is their hidden state, often referred to as the memory state. This hidden state stores important information from prior inputs in the sequence, enabling the network to maintain continuity and context. By sharing the same parameters across all steps in a sequence, RNNs reduce the number of parameters, making them more efficient than traditional neural networks for handling sequential tasks.

In simpler terms, RNNs work by applying the same network to each item in a sequence while preserving and passing along relevant information. This ability to learn temporal dependencies makes RNNs particularly effective for tasks like language modeling, time series forecasting, and other applications involving sequential or contextual data.

This makes them highly valuable in network intrusion prediction. RNNs can identify subtle anomalies or irregularities in network behavior that may indicate potential security threats.

Deep Belief Networks (DBNs)

Deep Belief Networks (DBNs) are advanced deep learning models built from stacks of Restricted Boltzmann Machines (RBMs). Each RBM consists of a visible layer that interacts with input data and a hidden layer that learns high-level features, with connections existing only between layers. The hierarchical, layers-wise training allows DBNs to extract complex patterns and model intricate relationships in data [41].

In network intrusion prediction, DBNs are highly effective due to their ability to uncover hidden patterns in network traffic data. By leveraging their energy-based learning, DBNs can detect subtle anomalies indicative of potential intrusions, enabling accurate and proactive threat detection. Their generative capabilities also allow them to simulate and analyze unusual traffic patterns, further enhancing intrusion prevention systems.

Long Short-Term Memory (LSTMs)

Long Short-Term Memory (LSTM) networks, a type of Recurrent Neural Network (RNN) introduced by Sepp Hochreiter and Jürgen Schmidhuber in 1977 [42], are highly effective at processing sequential data. Unlike traditional models that work with single data points, LSTMs can handle multiple sequences simultaneously, making them particularly suitable for analyzing network logs, where temporal dependencies and patterns are crucial [43; 44].

LSTMs were created to overcome the main weakness of traditional RNNs, their difficulty in remembering information over long sequences. They achieve this by retaining information over extended periods, effectively allowing data from earlier in the sequence to influence predictions or decisions later in the process. This is critical when analyzing network logs, as anomalies or patterns may span across a long sequence of events.

The architecture of an LSTM consists of a chain of repeating neural network modules, where information flows between them. Using structures called gates, LSTMs can selectively retain or discard information. These gates are controlled by sigmoid layers and point-wise multiplication, enabling fine-tuned regulation of the information flow. This flexibility enables LSTMs to learn complex, time-dependent patterns in network logs, such as detecting anomalies, identifying intrusion attempts, and predicting future network behavior.

When applied to network intrusion detection, LSTMs can analyze sequences of network logs to detect irregularities that might signal potential security threats. By leveraging their ability to capture long-term dependencies and adaptively manage information flow, LSTMs are a powerful tool for enhancing cybersecurity measures. However, their advanced capabilities come with increased computational complexity and resource requirements.

Autoencoders (AEs)

An autoencoder is a sophisticated neural network architecture designed to learn efficient data representations through a process of compression and reconstruction. At its core, the autoencoder operates on a simple yet powerful principle: it attempts to copy its input to its output, but in doing so, it's forced to learn a compressed, compact representation of the data in between [45]. The network typically consists of two main components:

- **An encoder**: which compresses the input data into a reduced-dimensional representation (latent space)
- **A decoder**: which attempts to reconstruct the original input from this compressed representation

The key innovation is the bottleneck layer (latent space), where data is reduced to its most fundamental characteristics. During training, the autoencoder learns to minimize the difference between the original input and its reconstruction, effectively discovering the most important data features.

Unlike supervised learning, autoencoders use unsupervised techniques to learn from the data itself. They identify underlying patterns without explicit labeling, creating a meaningful, compressed representation of the input.

For instance, in a corporate network, an autoencoder could learn the typical communication patterns between servers, user devices, and external services. Any significant deviation, such as an unexpected data transfer, unusual connection timing, or communication with unknown addresses, would trigger an alert for further investigation.

Compared to traditional machine learning approaches, autoencoders offer a more nuanced, context-aware method of detecting potential network intrusions by understanding the inherent structure of network data rather than relying on predefined rules or signatures.

2.3 Used Tools

The development of the proposed system required the integration of several software tools and platforms, selected for their reliability and suitability in deep learning-based intrusion detection. This subsection outlines the main technologies employed, including programming frameworks, data processing utilities, databases, and deployment environments, highlighting their specific role in supporting the system architecture.

Python

Python is an interpreted, high-level programming language recognized for its clear syntax, ease of use, and adaptability. It accommodates various programming paradigms such as procedural, object-oriented, and functional approaches. It is widely used in web development, data analysis, artificial intelligence, automation, and more. Pythons extensive standard library and large community contribute to its popularity and ease of use [46].

Scikit-Learn

Open-source Python library that provides a comprehensive collection of machine learning algorithms and data analysis tools. It supports tasks such as classification, regression, clustering, dimensionality reduction, and model evaluation, offering efficient implementations of widely used methods through a consistent interface. Its integration with libraries such as NumPy [47], SciPy [48], and Pandas [49] makes it a standard framework for developing and benchmarking machine learning models in both research and applied domains [50].

Tensorflow

Widely adopted by data scientists, developers, and educators, TensorFlow is an open-source machine learning platform that uses data flow graphs. In these graphs, the nodes denote mathematical operations, and the edges transmit multidimensional data arrays, commonly referred to as tensors, between them. This flexible design enables machine learning models to be built as networks of connected operations. TensorFlow supports execution on CPUs, GPUs, and TPUs across devices ranging from smartphones to servers, without requiring code changes. This consistency allows teams from diverse backgrounds to collaborate efficiently using the same tools. Originally created by the Google Brain Team for research in machine learning and deep neural networks, TensorFlow is versatile enough to be applied in many other fields as well [51].

CICFlowMeter

CICFlowMeter is a tool for generating and analyzing network traffic flows. It creates bidirectional flows, where the direction of the first packet defines the forward (source to destination) and backward (destination to source) paths. This allows for the calculation of over 80 statistical features, such as duration, packet count, byte count, and packet length, separately for both directions, providing detailed insights into network behavior [52].

Redis

Redis (REmote DIctionary Server) is an open-source, in-memory NoSQL key-value store, commonly used as a fast-response database or caching layer for applications [53].

InfluxDB

InfluxDB is an open-source time-series database platform built to efficiently handle the ingestion, storage, retrieval, and visualization of time-series data. It performs exceptionally well in scenarios involving high-volume, high-speed data streams, providing robust query functions, automated downsampling, and efficient data compression. [54].

Docker

Docker is a software platform designed to simplify the building, testing, and deployment of applications. It packages applications into standardized units called containers, which include all necessary components, such as code, libraries, system tools, and runtime. This ensures consistent performance across different environments and makes it easy to deploy and scale applications efficiently [55].

Docker Compose

Docker Compose is a tool for defining and running multi-container applications. It is the key to unlocking a streamlined and efficient development and deployment experience [56].

Ubuntu Linux

Ubuntu is a free and open-source Linux-based operating system developed by Canonical Ltd. It is based on Debian and designed for ease of use, security, and regular updates. Ubuntu is widely used on desktops, servers, and cloud platforms, and it comes with a wide range of pre-installed software. Its user-friendly interface and strong community support make it a popular choice for both beginners and experienced Linux users [57].

Kali Linux

Kali Linux, first released in 2013, is a widely used Debian-based Linux distribution tailored for penetration testing and digital forensics. It is developed, maintained, and funded by Offensive Security, a well-known cybersecurity company. Kali is the successor to BackTrack, an earlier Linux distribution also focused on security testing, and it comes preloaded with numerous tools designed for ethical hacking, vulnerability assessment, and network analysis [58].

2.4 Summary

This chapter provides the theoretical foundation required to facilitate the understanding of the proposed Network Intrusion Prediction System. It went over the basics of cybersecurity and intrusion detection, discussed the many forms of network traffic and attacks, and looked at the fundamentals of machine learning and deep learning models that are important for the task.

Chapter 3

State of the Art

This chapter provides an overview of the current state of the art in addressing the problem of network attack prediction. It explores not only the methodologies employed in building systems capable of detecting or predicting such attacks, but also the evaluation strategies and datasets used to train and test AI models in this domain. At last, the chapter discusses the research hypothesis, outlining the central aim of this dissertation.

3.1 Datasets for Intrusion Detection

An analysis of common data sets used in network intrusion detection reveals a significant reliance on datasets including **KDD 99** [59], **DARPA 2000** [60], **UNSW-NB15** [61], and **CIC-IDS2017** [7]. However, the use of these datasets, especially the older ones like KDD 99 and DARPA 2000, raises questions about their applicability to evolving cybersecurity threats [8; 34; 62].

The search for suitable datasets for intrusion detection research remains a challenge. Issues include concerns about data privacy, obtaining approval from data owners, the scope of evaluation datasets, and discrepancies between them. Additionally, documentation, understanding, and labeling of data within the datasets are important considerations.

To address the needs of intrusion detection research, various datasets have been created from simulated networks. This method allows researchers to replicate specific attack scenarios and collect data in controlled environments. However, generalizing findings from simulated environments to real-world scenarios remains a challenge.

The **SABU** [63] dataset (an alert-sharing platform) is a promising development, providing real-world intrusion alerts collected from various intrusion detection systems across different organizations. The SABU dataset offers valuable opportunities to investigate realistic attack scenarios and develop alert correlation techniques [34].

The selection of the best dataset for future work in predicting network intrusion is strongly influenced by the specific research objectives. However, we can draw certain conclusions based on the analysis of famous datasets and their limitations:

- Older datasets, such as KDD 99 and DARPA 2000, should generally be avoided due to their age and limited relevance to modern network environments. Their outdated attack profiles and simplified traffic patterns no longer reflect current threats [34; 8; 62].
- More recent datasets, such as the UNSW-NB15 and CIC-IDS2017, include a greater variety of attack types and are seen to be more representative of modern threats. However, even these datasets may have limitations, resulting in the need for more extensive datasets in real time to accommodate the nature of the rapidly evolving cyberattacks [64].
- Simulated datasets, such as **FLNET2023** [65] and **5G-NIDD** [66], tend to yield good results with deep learning methods, however, their ability to mirror real-world intrusion scenarios needs further validation [64].
- The SABU dataset emerges as a particularly promising option, as it comprises real-world traffic collected from multiple organizations. This makes it potentially more reflective of practical intrusion detection challenges. Nevertheless, its use requires extensive preprocessing, which may complicate its application in deep learning pipelines [34].

For future research, the ideal dataset would address the limitations of existing ones. Specifically, such a dataset should be:

- Recent and up-to-date, including the latest attack types.
- Comprehensive, covering a variety of attack scenarios and network environments.
- Publicly available, to foster collaboration and research reproducibility.
- Well-documented, with clear information about attack types, characteristics, and collection methods.
- Ethically sound, respecting data privacy and legal considerations.

Developing and utilizing a dataset that meets these criteria will enable the creation of more effective and robust intrusion detection systems.

It is important to remember that no dataset is perfect. The choice of the most suitable dataset will always depend on the specific research context and the questions being addressed.

3.1.1 The CIC-IDS2017 Dataset

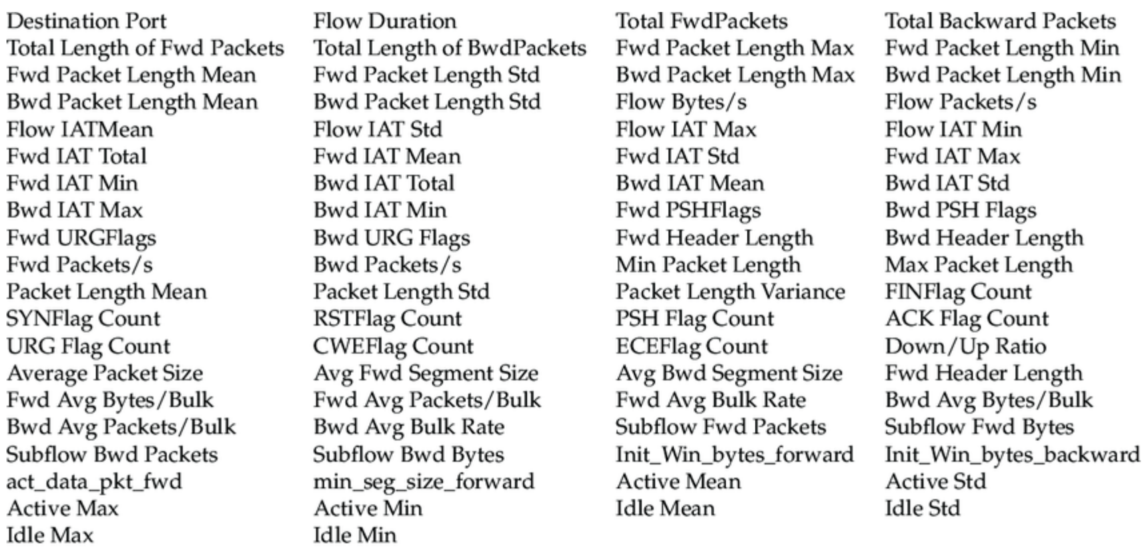
The CIC-IDS2017 dataset is distinguished by several unique qualities that enhance its utility for intrusion detection research. It offers a comprehensive representation of real-world network traffic, representing a variety of attack scenarios, which is crucial for developing effective intrusion detection systems.

One of the main advantages of the dataset is the richness of its features. It includes five full days of network traffic analyzed to extract more than 80 features from network flow data. These features capture various aspects of traffic, including packet length, duration, protocol type, flow bytes/s, flow packets/s, and other statistical measures. This vast feature set provides a detailed perspective of network activity, enabling the creation of advanced models capable of accurately distinguishing between hostile and benign traffic patterns. The following are the key aspects that separate this dataset from others.

Number of Features

- The original dataset contains 79 features (after removing the non-valuable ones) extracted from network traffic flows. However, after data cleaning and pre-processing, the number of features used for machine learning may be reduced. For example, one study removed some features due to miscalculation or redundancy, including 'Bwd PSH Flags', 'Bwd URG Flags', and several bulk-related features, as well as timestamp and flow identifier, using 71 features after PCA [67; 68]. Another study removed features such as the IP addresses and ports and also removed features that were always null [68].

The original features are:



Destination Port	Flow Duration	Total FwdPackets	Total Backward Packets
Total Length of Fwd Packets	Total Length of BwdPackets	Fwd Packet Length Max	Fwd Packet Length Min
Fwd Packet Length Mean	Fwd Packet Length Std	Bwd Packet Length Max	Bwd Packet Length Min
Bwd Packet Length Mean	Bwd Packet Length Std	Flow Bytes/s	Flow Packets/s
Flow IATMean	Flow IAT Std	Flow IAT Max	Flow IAT Min
Fwd IAT Total	Fwd IAT Mean	Fwd IAT Std	Fwd IAT Max
Fwd IAT Min	Bwd IAT Total	Bwd IAT Mean	Bwd IAT Std
Bwd IAT Max	Bwd IAT Min	Fwd PSHFlags	Bwd PSH Flags
Fwd URGFlags	Bwd URG Flags	Fwd Header Length	Bwd Header Length
Fwd Packets/s	Bwd Packets/s	Min Packet Length	Max Packet Length
Packet Length Mean	Packet Length Std	Packet Length Variance	FINFlag Count
SYNFlag Count	RSTFlag Count	PSH Flag Count	ACK Flag Count
URG Flag Count	CWEFlag Count	ECEFlag Count	Down/Up Ratio
Average Packet Size	Avg Fwd Segment Size	Avg Bwd Segment Size	Fwd Header Length
Fwd Avg Bytes/Bulk	Fwd Avg Packets/Bulk	Fwd Avg Bulk Rate	Bwd Avg Bytes/Bulk
Bwd Avg Packets/Bulk	Bwd Avg Bulk Rate	Subflow Fwd Packets	Subflow Fwd Bytes
Subflow Bwd Packets	Subflow Bwd Bytes	Init_Win_bytes_forward	Init_Win_bytes_backward
act_data_pkt_fwd	min_seg_size_forward	Active Mean	Active Std
Active Max	Active Min	Idle Mean	Idle Std
Idle Max	Idle Min		

Figure 3.1: CIC-IDS2017 features.

Number of Instances

- The dataset initially contains a total of 2,830,743 instances of network traffic. However, after cleaning, the number of instances is reduced. One study removed instances with empty features, 'NaN', or 'Infinity' values [68]. Another study, using a corrected dataset called LYCOS-IDS2017 [69], had 1,837,500 instances after dropping packets from Thursday afternoon and addressing TCP termination issues [68]. A subset of the dataset used for machine learning had 440,632 instances for training, 220,312 for cross-validation, and 220,312 for testing.

Diverse Attack Scenarios

- The dataset has 15 classes, consisting of one class for normal traffic (BENIGN) and 14 classes for different types of attacks. For some machine learning tasks, the classes may be reduced to a binary classification problem, with one class for normal traffic and another class combining all attacks. Another study focused on a 5 class output [67].
- The dataset includes multiple types of attacks such as Denial-of-Service (DoS), Distributed DoS (DDoS), web attacks, and brute force attacks, providing a rich source for training and testing intrusion detection models [70].
- It captures both benign and malicious traffic, allowing for balanced evaluations of detection algorithms [70].

The original classes are:

- BENIGN
- DoS Hulk
- DDoS
- PortScan
- DoS GoldenEye
- FTP-Patator
- DoS slowloris
- DoS Slowhttptest
- SSH-Patator
- Bot
- Brute Force

- XSS
- Infiltration
- Sql Injection
- Heartbleed

Number of Instances Per Class

The dataset is highly imbalanced, meaning that the number of instances varies significantly between classes.

- The BENIGN class makes up the majority of the dataset, with 2,273,097 instances in the original CIC-IDS2017 dataset.
- Some attack classes have a large number of instances, such as DDoS with 128,027 instances and DoS Hulk with 231,073 instances.
- Other attack classes are represented by very few instances, such as Heartbleed with 11 instances.

Data Format and Accessibility

- CIC-IDS2017 is available in both raw data and flow-based features in CSV format, facilitating easier access and analysis for researchers [68].
- The dataset is labeled, which is essential for supervised learning approaches in machine learning [71].

Enhanced Feature Representation

- The dataset addresses limitations found in older datasets, such as KDD-Cup99, by providing a more accurate representation of network threats, thus improving the efficacy of machine learning algorithms [72].
- It includes a significant number of unique features, which can be optimized for better detection performance [71].

Dataset Composition

- **Rich Feature Set**: Initially comprising 83 features, the dataset captures essential network traffic characteristics, which can be reduced to 10 key features for efficient processing [73].

In contrast, while CIC-IDS2017 offers a robust framework for intrusion detection, some researchers argue that the dataset may still contain inherent biases or limitations that could affect the generalizability of models trained on it, necessitating further refinement and validation [68].

The CIC-IDS2017 dataset was chosen for this study because it accurately and comprehensively represents modern network traffic. It has a wide range of attack types and a lot of features, which makes it a great tool for training and testing deep learning models that are meant to predict network intrusions. Also, turning raw PCAP files into dataset rows that follow the same feature format as the dataset is relatively straightforward. This facilitates the integration of the dataset to a real-world deep learning model pipeline, making consistent inferences on new network traffic.

3.2 Related Work Comparison

Advances in computational learning techniques have had a big impact on how intrusion detection has changed over time. Traditional Machine Learning techniques depend on manual feature engineering and have demonstrated constraints in adjusting to changing attack patterns. Deep Learning methods, on the other hand, can automatically extract features and better model complex behaviors in network traffic. This section compares ML and DL methods for predicting network attacks, pointing out their pros and cons and how well they work for modern intrusion detection tasks.

3.2.1 Comparison of ML Methods and DL Architectures

The effectiveness of intrusion detection systems depends significantly on the choice of the learning approach. Traditional Machine Learning (ML) methods rely on manually engineered features and have been widely applied in earlier studies, while Deep Learning (DL) architectures leverage automated feature extraction and hierarchical pattern recognition. This subsection compares representative ML techniques with DL approaches, highlighting their respective strengths, limitations, and suitability for network attack prediction.

Traditional Machine Learning (ML) Methods

Many traditional ML techniques are used for network intrusion detection, these include K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Naive Bayes (NB), Multi-Layer Perceptron (MLP) and Decision Trees (DT). Ensemble methods such as Random Forest, AdaBoost, and XGBoost are also used [74]. But when talking of traditional ML methods working alone in such complex task there are several shortcomings [64; 6]. A significant drawback is their reliance on manual feature engineering [75; 8]. This is a process that takes time, and a high level of expertise to select and transform relevant features from raw network data. We also need to have in mind that this task complexity may, with great probability, lead to errors (or low performance) in a systematic way. The effectiveness of these models are heavily dependent on the quantity of these engineered features, and they may fail to capture the complex and nuanced patterns characteristics of modern cyberattacks [64]. Additionally, ML algorithms often struggle to handle large volumes of high-dimensional data effectively. They also may be inclined to overfitting, which means that they will perform well on training data but poorly on unseen data.

Furthermore, many ML models generate high positive rates, which can overwhelm security analysts and lead to alert fatigue. This is because traditional ML models tend to treat all features with equal importance and may not detect complex interactions within the data [64]. ML-based systems are often better suited to static networks, and may struggle with scalability in dynamic networks, where the network configurations may change rapidly. ML methods may also struggle to adapt to new attack vectors or to recognize zero day attacks, as their effectiveness is tied to their learned feature sets [34].

Advantages of Deep Learning (DL) in Network Attack Prediction

DL, as a subfield of ML, addresses many of the shortcomings of traditional approaches by using neural networks with multiple layers, allowing for more abstract feature extraction [75; 76; 62]. DL models can automatically learn complex patterns and relationships within network traffic data, eliminating the need for manual feature engineering. This automatic feature extraction enables DL models to adapt to evolving attack vectors and detect zero-day attacks more efficiently than traditional ML methods.

DL architectures like CNNs can process spatial data, making them effective at identifying patterns in network traffic that has been converted into an image. CNNs have also demonstrated a capacity to identify multiple types of network attacks with high accuracy. RNNs, especially LSTMs and Gated Recurrent Units (GRU), are capable of capturing temporal dependencies in sequential data such as network traffic. They are particularly effective for intrusion detection because they can analyze the context of network events over time, helping detect multi-stage attacks. DBNs also provide benefits by being able to extract sophisticated features and identify complex patterns, also showing consistency in their performance and training time [77].

DL models also allow for end-to-end learning, processing raw data and extracting complex features that are directly relevant to the classification task. This makes them particularly effective in detecting novel attack patterns that may not have been seen before. Another key advantage is DL's capacity for predictive analysis, making it a powerful tool to use for more proactive security measures [6].

DL methods tend to exhibit lower false positives rates, thereby reducing the risk of alert fatigue [75]. Furthermore, DL models can achieve higher accuracy and detection rates than traditional ML methods, particularly when trained on large datasets, showing better performance in complex datasets [77].

The Superiority of DL Over ML

The key difference between DL and ML lies in their capacity for feature extraction. ML models are dependent on hand-crafted features which require domain expertise and are prone to bias, whereas DL models extract these features automatically from raw data. This provides a significant advantage in the dynamic environment of network security where new attacks and network traffic patterns are continually emerging [64; 75; 76; 8].

DL methods have also been shown to achieve better results when working with large amounts of data. This is because the complex architecture of neural networks needs sufficient data to learn the various patterns effectively. DL methods, particularly hierarchical models, are also adept at reducing false positives, as they can effectively filter out non-attack-related traffic before passing on information for deeper analysis. This is important as it can greatly reduce the cost of manual verification [77]. DL models have also been shown to reduce false positives versus non-hierarchical approaches [75]. Hybrid approaches, which integrate DL with traditional ML, combine the strengths of both methods and can achieve even greater performance [64; 8].

Hybrid and Ensemble Methods

The effectiveness of Hybrid and Ensemble Methods compared to Deep Learning (DL) and Machine Learning (ML) in detecting zero-day attacks in Intrusion Detection Systems (IDS) is also a important area of study. Hybrid approaches, which combine the strengths of DL and ensemble techniques, have shown significant promise in enhancing detection accuracy and reducing false positives.

Hybrid models leverage both DL (e.g., CNNs, LSTMs) and ensemble methods (e.g., Random Forest, Gradient Boosting) to capture complex patterns in network traffic, these models have achieved high detection accuracy (up to 97.8%) and low false-positive rates (0.022), outperforming traditional methods [78]. The combination of multiple classifiers enhances resilience against diverse attack types, including zero-day exploits [79].

Model Evaluation and Comparative Performance

As previously said, performance of both DL and ML models depends heavily on the quality and quantity of the data used for training and evaluation metrics such as precision, recall, F1-score, and false positive rate are used to measure the effectiveness of these models [77]. As such, it is important to use datasets with high quality and a good balance between normal and attack traffic to ensure the models are correctly evaluated [64].

While traditional ML methods have played a role in network attack detection, DL represents a more powerful and versatile approach that has consistently shown superior performance. Its ability to automatically extract features, adapt to new threats, and reduce false positives makes it the best option for modern network security environments. Although DL models may be more complex and require more data, their enhanced performance and proactive capabilities make them a vital tool in the fight against cyber threats.

Several research works have investigated machine learning and deep learning approaches for network attack prediction, with studies such as [80; 81; 82] providing comprehensive reviews of previous methods applied to this domain. The comparative analysis of these approaches reveals a clear trend: while traditional ML techniques such as SVM, K-NN, Naive Bayes, Random Forest, and ANN achieve strong performance in certain scenarios, their results are often inconsistent across datasets and attack types, with detection rates dropping significantly for rare or complex categories (e.g., U2R and R2L in KDD99). By contrast, deep learning architectures, particularly CNNs, RNN variants (LSTM, BLSTM), and hybrid models, consistently demonstrate higher accuracy, improved recall, and lower false alarm rates, with several studies reporting accuracies above 98-99% on modern datasets such as CIC-IDS2017 and UNSW-NB15. These findings indicate that DL-based methods provide superior generalization and robustness for contemporary intrusion detection, although their complexity and computational requirements remain greater than those of conventional ML models. A structured overview of these approaches is presented in Table 3.1.

Table 3.1: Comparative Overview of Machine and Deep Learning Approaches for Intrusion Detection

Article	ML/DL Architecture	Dataset	Performance Metrics (%)
[83]	SVM	KDD99	Detection Rate (DR): DoS 91.6, Probe 35.65, U2R 12.0, R2L 22.0
[84]	LMRDT-SVM	NSL-KDD	Accuracy: 99.31, DR: 99.20

Continued on next page...

Article	**ML/DL Architecture**	**Dataset**	**Performance Metrics (%)**
[85]	K-NN	KDD99	Accuracy: 99.89
[86]	Naive Bayes	NSL-KDD, UNSW-NB15, CIC-IDS2017	NSL-KDD: Accuracy 97.0 UNSW-NB15: Accuracy 86.9 CIC-IDS2017: Accuracy 98.59
[87]	Random Forest	UNSW-NB15	Accuracy: 97.49
[88]	ANN	KDD-99	DR: DoS 99.93, U2R 96.51, R2L 92.54, Probe 98.7
[89]	CNN	KDD99	Accuracy: 99.95, FAR: 0.022
[90]	RNN-LSTM	CAIDA DDoS 2007	Accuracy: 96.0
[91]	RNN-BLSTM	CIC-IDS2017	Accuracy: 98.48
[92]	RNN	NSL-KDD	Accuracy: Training 99.81, Testing 83.28
[93]	Autoencoder (AE)	KDD CUP 99	Accuracy: 94.71
[94]	CNN	UNSW-NB15, CIC-IDS2017	UNSW-NB15 (BC): Recall 99.74 UNSW-NB15 (MC): Recall 96.54 CIC-IDS2017: Recall 99.85
[95]	CNN, LSTM	ISCX2012	Accuracy: 99.69, Recall: 96.91, FPR: 0.22
[96]	CNN, RNN, LSTM, GRU	NSL-KDD	CNN+LSTM: Accuracy 99.7, Precision 99.9, Recall 99.6, F1 99.8 CNN+GRU: Accuracy 98.1, Precision 99.9, Recall 97.6, F1 98.8 CNN+RNN: Accuracy 97.3, Precision 100, Recall 96.7, F1 98.3

These studies demonstrate the effectiveness of various models in achieving high accuracy for network attack prediction.

Based on the comparative analysis of existing approaches, Long Short-Term Memory (LSTM) networks stand out as a particularly suitable architecture for network attack prediction. Their ability to capture long-term temporal dependencies makes them highly effective in modeling sequential data such as network traffic flows, where contextual relationships between packets are crucial for accurate attack identification. While other architectures like CNNs or hybrid CNN-RNN models excel in spatial feature extraction or combined representations, LSTMs are uniquely capable of learning time-dependent behavioral patterns that often precede intrusion events [44; 77]. Previous studies employing LSTM-based models have demonstrated excellent detection accuracy and robustness across various datasets [90; 91; 95; 96]; however, only a limited number have applied this architecture specifically to the

CIC-IDS2017 dataset. Given the datasets realistic traffic composition and diverse attack scenarios, this combination presents a promising research direction. Therefore, this dissertation adopts an LSTM architecture trained on CIC-IDS2017 to evaluate its suitability and effectiveness for multiclass network intrusion prediction.

3.3 Evaluation Strategies

Evaluating deep learning (DL) models in intrusion detection systems (IDS) involves various methodologies that assess their performance, robustness, and effectiveness against cyber threats. The evaluation typically focuses on accuracy, detection rates, and the ability to minimize false positives.

Performance Metrics

- **Accuracy**: Measures the proportion of true results among the total cases examined. High accuracy indicates effective detection capabilities [97].
- **Detection Rate**: This metric assesses the percentage of actual attacks that are correctly identified by the model, crucial for evaluating the model's effectiveness in real-world scenarios [97].
- **False Alarm Rate**: This indicates the frequency of false positives, where benign traffic is incorrectly classified as malicious. Lower rates are preferable for operational efficiency [97].

Model Robustness

- **Adversarial Testing**: Evaluating models under adversarial conditions helps determine their resilience against attacks designed to deceive the system. Studies have shown that while DL models improve detection accuracy, they can be vulnerable to adversarial samples [98].

While these evaluation methods highlight the strengths of DL models in intrusion detection, it is essential to consider the limitations, such as the potential for overfitting and the need for continuous updates to adapt to evolving threats.

3.4 Deployment and Real-World Challenges

The deployment of deep learning (DL) and machine learning (ML) models in intrusion detection systems (IDS) faces several real-world challenges that can hinder their effectiveness. These challenges stem from the complexity of cyber threats, the need for extensive data, and the interpretability of models.

Data Requirements

The efficacy of ML and DL models in intrusion detection is predominantly conditioned upon the quality and volume of the training data utilized. However there are some problems with the nature and availability of this data.

- **Large-scale Datasets**: ML and DL models require vast amounts of labeled data for training, which can be difficult to obtain in real-world scenarios [99].
- **Data Complexity**: The intricate nature of network traffic and evolving attack patterns complicates the training process, often leading to high false positive rates [100].

Model Interpretability

Deep learning models often have high detection accuracy, but it's still a big problem to understand how they make decisions, especially when it comes to security.

- **Opaque Decision-Making**: Many DL models operate as "black boxes", making it challenging for security analysts to understand how decisions are made, which can hinder trust and adoption [101].
- **Adversarial Vulnerability**: DL models can be susceptible to adversarial attacks, where slight modifications to input data can lead to incorrect classifications [99].

Computational Demands

The deployment of advanced ML and DL techniques also introduces substantial computational challenges, both during model training and real-time inference.

- **Resource Intensive**: The computational power required for training and deploying these models can be prohibitive, especially for smaller organizations [101].

3.5 Summary

In this chapter, a comprehensive review of the current state of the art in network intrusion detection and prediction was conducted. The analysis began by evaluating widely used datasets in cybersecurity research. While legacy datasets like KDD 99 and DARPA 2000 have historical significance, their lack of alignment with modern threat landscapes makes them unsuitable for contemporary use. In contrast, the CIC-IDS2017 dataset, selected for this study, offers a robust representation of real-world attack vectors and a well-labeled, feature-rich format conducive to deep learning experimentation.

The chapter further examined a range of traditional machine learning (ML) techniques and advanced deep learning (DL) architectures. ML approaches such as SVM, KNN, and Random Forest have been effective in certain contexts, but often fall short in handling large-scale, high-dimensional data and require extensive manual feature engineering. In contrast, DL methods, particularly architectures like MLPs, CNNs, RNNs, LSTMs, and hybrid models, have demonstrated superior performance due to their capacity for automatic feature extraction, temporal pattern learning, and reduced false-positive rates.

The deployment of DL-based systems, however, comes with challenges related to computational resources, data labeling, and model interpretability. Nevertheless, evaluation strategies show that DL architectures consistently outperform traditional methods in both detection accuracy and robustness, especially in dynamic and complex network environments.

Chapter 4

Model Implementation

This chapter details the development of the predictive model used in this study. It focuses on the practical implementation of the solution using deep learning techniques, building on the theoretical background and research problem introduced in the previous chapters. The process is divided into two major components: 1) data preparation; and 2) model development.

The data preparation stage encompasses several steps aimed at transforming the raw CIC-IDS2017 dataset into a structured and balanced format suitable for deep learning. First, the original CSV files were concatenated into a single dataset, after which irrelevant or redundant features were removed. Next, data cleaning procedures were applied, including the elimination of zero-variance columns, duplicate rows, and highly correlated features. To better understand the dataset, visualizations of traffic distributions were generated, followed by strategies to address class imbalance. A train-validation-test split was then performed to ensure robust evaluation, and a quantile transformation was applied to normalize feature distributions. Finally, categorical labels were one-hot encoded, and the final dataset shapes were established for subsequent modeling.

The model construction stage describes the design, training, and evaluation of the predictive deep learning model. Specifically, an LSTM-based architecture was built, including the selection of layer types, neuron counts, activation functions, and dropout strategies to mitigate overfitting. The model was compiled with an appropriate loss function and optimizer, and training was conducted with early stopping to prevent unnecessary epochs. Evaluation was performed on a held-out test set, using not only accuracy but also precision, recall, F1-score, and confusion matrices to assess class-level performance. Lastly, the trained model was saved for reuse, ensuring reproducibility and portability in future experiments.

Together, these two components, data preprocessing and model development, form the foundation of the proposed intrusion prediction system, enabling accurate and scalable multiclass attack detection.

4.1 Data Preparation

This section outlines the series of transformations applied to prepare the data for training the LSTM model. The preprocessing pipeline includes steps ranging from feature reduction to feature selection, ensuring that the input data is both relevant and well-structured for effective model training.

4.1.1 Data Concatenation

Data was obtained in the Canadian Institute for Cybersecurity [102] platform in the form of various Comma-Separated Values (CSV) files. Since data separation across multiple files was unnecessary for model training, all files were concatenated into a single CSV. After concatenation the dataset shape is (2830743, 79).

4.1.2 Data Cleaning and Feature Selection

Data Cleaning is a relatively simple process, however Feature Selection is not. Let us first analyze the motives why a column (a feature) should be removed from the dataset.

Features to consider removing

- **Highly Correlated Features**: Feature that are highly correlated e.g., `Fwd Packet Length Max` and `Fwd Packet Length Mean`) may provide redundant information. The usage of a correlation matrix to identify these is a good method.

- **Statistical Aggregates**: Detailed statistics (mean, max, min, std) for packet lengths, I considered keeping only a few representative aggregates. For example keeping Mean or Std and drop Max and Min if their patterns do not add unique predictive power.

- **Low Variance Features**: Features that show little or no variation across the dataset do not contribute to distinguishing between classes.

For the initial data cleaning stage, all negative values were adjusted to ensure that every feature had a minimum value of zero. As mentioned earlier, columns with zero variance were removed since they provide no discriminative power for classification. Finally, duplicate rows and highly correlated features, identified as having identical or redundant values, were eliminated to reduce redundancy and improve model efficiency.

Dropped Features and Dataset Shape after Data Cleaning

- **Dropped Zero Variance Columns**: `Bwd PSH Flags`, `Bwd URG Flags`, `Fwd Avg Bytes/Bulk`, `Fwd Avg Packets/Bulk`, `Fwd Avg Bulk Rate`, `Bwd Avg Bytes/Bulk`, `Bwd Avg Packets/Bulk`, `Bwd Avg Bulk Rate`.
- **Dropped Columns Pairs with Identical Values**: (`Total Fwd Packets`, `Subflow Fwd Packets`), (`Total Backward Packets`, `Subflow Bwd Packets`), (`Fwd PSH Flags`, `SYN Flag Count`), (`Fwd URG Flags`, `CWE Flag Count`), (`Fwd Header Length`, `Fwd Header Length.1`).
- **Dataset shape after dropped columns and rows**: (2520798, 66)

4.1.3 Data Visualization

Data visualization can help us identify what features are more likely to make the difference in the predicting task. It can also help us identify the fouls that the dataset has, for example class imbalance. A new features was created (Traffic type), for better visualization of malicious and normal traffic.

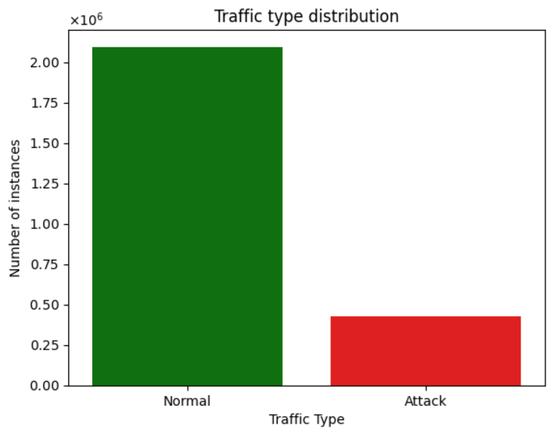


Figure 4.1: Plot distribution of Normal traffic and Attacks.

Figure 4.1 illustrates the distribution of network traffic types within the dataset. As we can see, the dataset is heavily imbalanced with Normal traffic significantly outnumbering Malicious traffic. While both under-sampling and over-sampling are viable methods to address class imbalance, I've chosen under-sampling not due to hardware constraints, but because synthetic data generated by over-sampling may not truly represent real-world scenarios [103]. This approach ensures that the model is trained only on genuine observations, potentially leading to more reliable performance in practical applications.

After under-sampling the shape of the dataset is (926612, 67). The distribution of network traffic types, within the dataset, after under-sampling is shown in Figure 4.2.

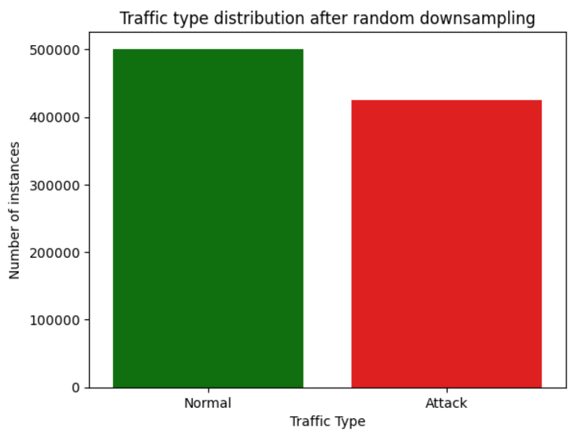


Figure 4.2: Plot distribution of Normal traffic and Attacks after under-sampling.

For further feature selection, a Random Forest classifier was used to estimate the importance of each feature for the classification task, leveraging its embedded feature selection mechanism and intrinsic metrics to rank features by importance [104]. Based on a critical visual analysis of the resulting feature-importance plot, a threshold of 0.001 was defined.

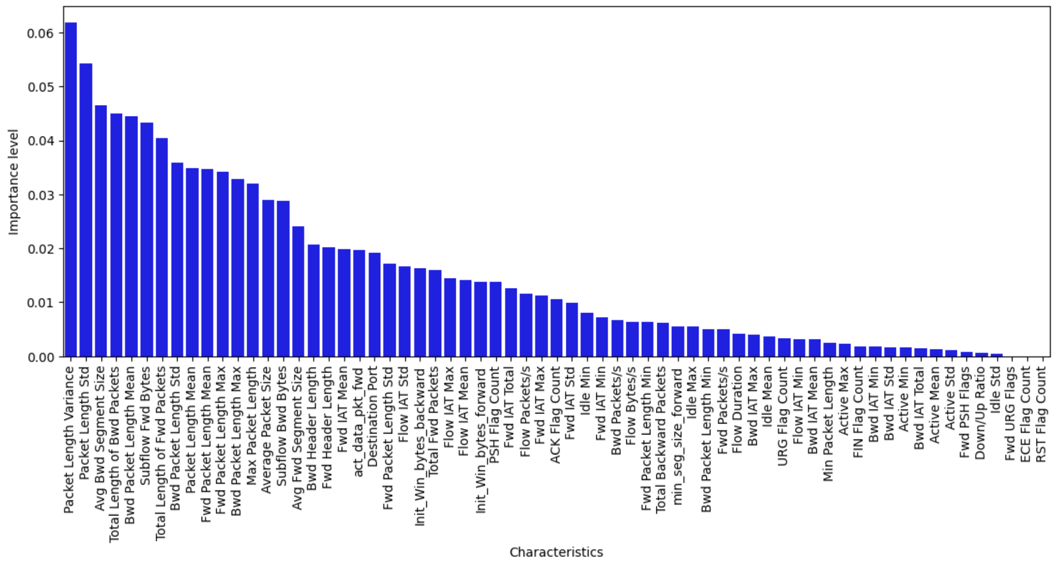


Figure 4.3: Feature Importance level for classification task.

Figure 4.3 presents the feature importance ranking obtained using a Random Forest Classifier. Points up the contribution of each attribute to the classification task, with features such as `Packet Length Variance` and `Packet Length Std` showing the highest relevance.

The features that were dropped (below the threshold of 0.001) were: `Fwd PSH`

`Flags`, `Down/Up Ratio`, `Idle Std`, `Fwd URG Flags`, `ECE Flag Count`, `RST Flag Count`. After this step, the dataset has the shape of (926612, 61).

The highly correlated features are also columns we can consider to drop. To analyze which pair of features are highly correlated, we displayed a correlation matrix (Figure 4.4). A correlation matrix is a table that shows the correlation coefficients between pairs of variables (features). Each value in the matrix indicates how strongly two variables are linearly related, with values ranging from -1 (perfect negative correlation) to +1 (perfect positive correlation). It's commonly used to identify redundant features, multicollinearity (when more than two independent variables are correlated [105]), or relationships that can impact model performance [106].

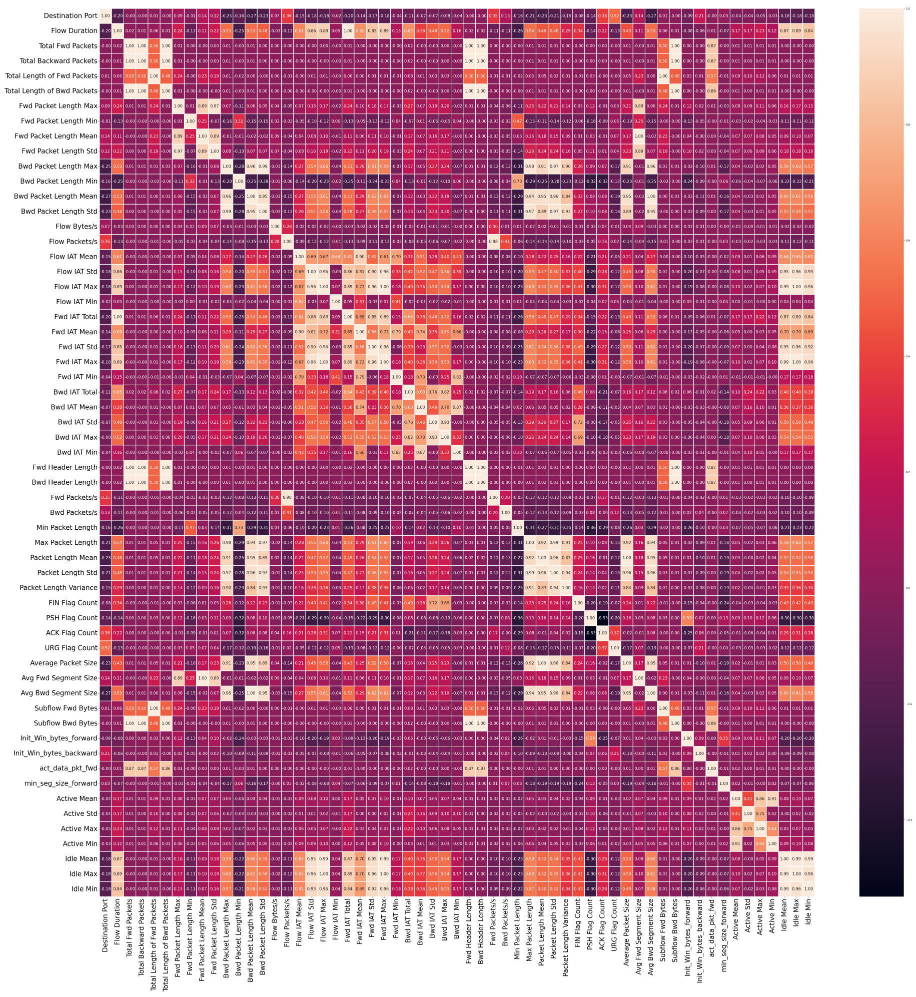


Figure 4.4: Features correlation matrix with highly correlated pairs.

Figure 4.4 presents the correlation matrix of all numerical features in the dataset. Each cell represents the correlation coefficient between two features, where lighter colors indicate stronger correlations, and darker tones represent weaker relationships. The diagonal line of bright cells corresponds to each features perfect correlation with itself. The matrix reveals several groups of features that are highly correlated, particularly among packet length and flow-related attributes, suggesting redundancy that can be addressed through feature reduction to improve model efficiency and prevent multicollinearity.

For the elimination of some features, we established a threshold of 0.95 to target these specific high-correlation groups without aggressively reducing the feature space. We also implemented the following rule (considering the importance matrix created with the Random Forest Classifier earlier):

- If a feature has a high correlation with a feature of great importance, the selected feature is dropped.
- If a feature has a high correlation with a feature of lower importance, the feature with lower importance is dropped.

These conditions can be formally expressed as follows.

Let f_i and f_j be two features, and let:

$$\rho(f_i, f_j) = \text{correlation}(f_i, f_j)$$

$$I(f_i), I(f_j) = \text{importance scores from the Random Forest Classifier.}$$

If:

$$|\rho(f_i, f_j)| > \tau$$

(where $\tau = 0.95$ is the correlation threshold), then:

$$f_k = \begin{cases} f_i, & \text{if } I(f_i) < I(f_j) \\ f_j, & \text{if } I(f_j) \leq I(f_i) \end{cases}$$

and the feature f_k is removed.

The process runs until all the highly correlated pairs (>0.95) disappear. After one iteration, the deleted features were: `Idle Mean`, `Flow Duration`, `Fwd Packets/s`, `Idle Max`, `Total Backward Packets`, `Idle Min`, `Fwd IAT Std`, `Fwd IAT Max`, `Flow IAT Max`, `Total Fwd Packets`, `Fwd Packet Length Std`, `Fwd Header Length`, `Bwd Header Length`, `Avg Fwd Segment Size`, `Subflow Bwd Bytes`, `Average Packet Size`, `Max Packet Length`, `Bwd Packet Length Max`, `Packet Length Mean`, `Bwd Packet Length Std`, `Total Length of Fwd Packets`, `Bwd Packet Length Mean`, and `Avg Bwd Segment Size`. 23 features were dropped leaving the dataset with the shape (926612, 38).

This correlation-based reduction step not only minimizes redundancy among features but also simplifies the models input space, improving training efficiency and potentially reducing overfitting. By retaining only the most informative and independent features, the resulting dataset becomes more compact and better suited for the deep learning model implemented in the subsequent stages.

Figure 4.5 presents the correlation matrix obtained after removing the highly correlated feature pairs. Similar to Figure 4.4, each cell represents the correlation coefficient between two features, where lighter colors denote stronger correlations and darker shades indicate weaker relationships. As observed, the remaining features exhibit considerably lower intercorrelation, confirming that the feature reduction process effectively minimized redundancy and preserved only the most independent variables for model training.

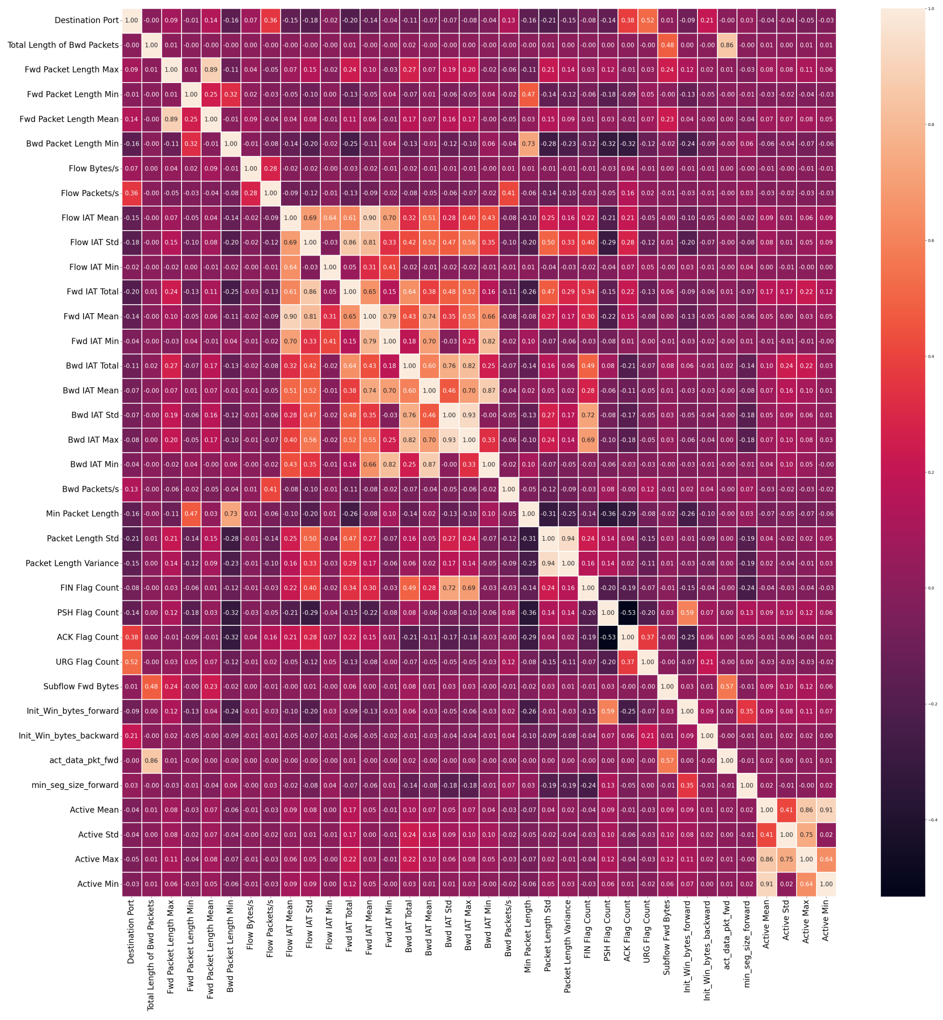


Figure 4.5: Features correlation matrix without highly correlated pairs.

4.1.4 Train Test Split

The dataset was initially divided into training (70%) and testing (30%) subsets, using stratified sampling to preserve class distribution. From the test subset, 20% of the original data (equivalent to 6% of the total dataset) was further separated to create a validation set. The final data distribution was therefore as follows:

- Training set: 70%
- Validation set: 6%
- Test set: 24%

4.1.5 Data Transformation

To normalize the input features and improve model convergence, a Quantile Transformer [107; 108; 109] was applied. This method transforms the distribution of each feature to follow a uniform or normal distribution by mapping the data to quantiles and applying a non-linear transformation. Unlike standard normalization techniques (e.g., z-score), the Quantile Transformer is robust to outliers and can significantly improve the handling of skewed or heavy-tailed distributions. This is particularly beneficial for LSTM networks, which are sensitive to the scale and distribution of input data, as it helps stabilize gradients and accelerate learning.

The transformation can be mathematically expressed as:

$$x' = G^{-1}\left(\widehat{F}_X(x)\right),$$

where $\widehat{F}_X$ denotes the empirical cumulative distribution function (ECDF) of the input data, and G^{-1} is the inverse cumulative distribution function (quantile function) of the target distribution.

In this study, the Quantile Transformer was used with default parameters, which map the data to a uniform distribution. Its important to note that the transformer should be fitted only on the training data to avoid data leakage.

4.1.6 Classes in Training Data

Plotting the classes in the training data we can see that the classes have a big imbalance.

Figure 4.6 illustrates the distribution of classes in the training dataset, clearly showing a pronounced imbalance, with certain attack categories being heavily underrepresented compared to normal traffic.

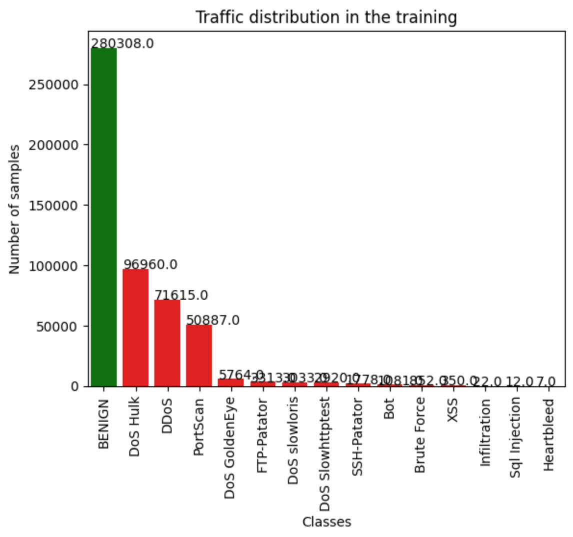


Figure 4.6: Classes distribution in training data.

To reduce the imbalance between the minority classes it was used SMOTE in the training and validation subsets. The Synthetic Minority Over-sampling Technique (SMOTE) is a widely used method in machine learning to address class imbalance in datasets, particularly in binary classification problems where one class (the minority) is underrepresented compared to the other (the majority).

After the oversampling the class distribution is represented in Figure 4.7.

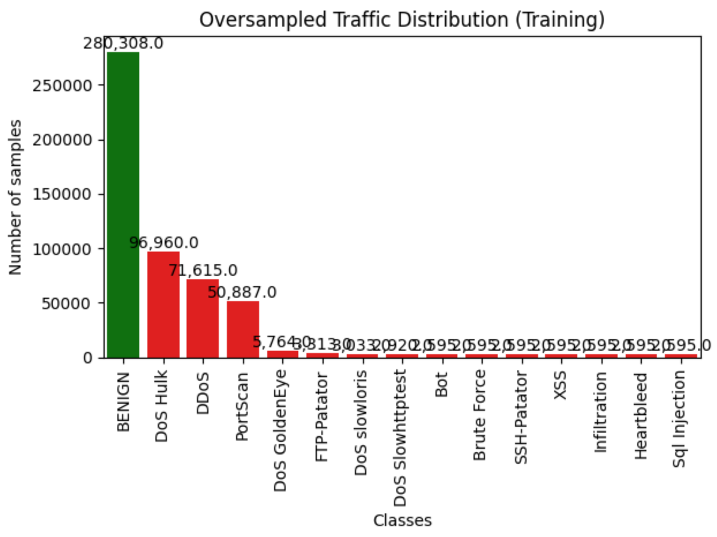


Figure 4.7: Classes distribution in training data after SMOTE oversampling.

SMOTE works by generating synthetic examples of the minority class rather than simply duplicating existing ones. This is achieved by selecting a minority class instance and introducing synthetic examples along the line segments joining any or all of the k minority class nearest neighbors. This approach helps the classifier to build larger decision regions that contain nearby minority class points, leading to better generalization [110; 111].

The same was done in the validation subset.

4.1.7 One-hot Encoding

Finally the classes where one-hot encoded. One-hot encoding transforms each unique category of a categorical variable into a new binary feature. For a variable with n distinct categories, this results in n binary features, each indicating the presence (1) or absence (0) of a specific category. This approach ensures that the model does not assume any ordinal relationship between categories, which could occur if categories were simply assigned arbitrary numerical values [112].

4.1.8 Final Subsets Shapes

After all data preprocessing the subsets have the following shapes:

- Training set: (532948, 36)
- Validation set: (129726, 36)
- Test set: (277984, 36)

4.2 Model Construction

This section describes the LSTM model's construction, training, and evaluation. It is utilized for the multiclass classification of network traffic. The model was created using a deep learning technique designed for sequential patterns and time-dependent features, building on the preprocessed and structured data. The design, which includes regularization techniques to reduce overfitting, was carefully selected in order to find a balance between model complexity and generalization capabilities. The accuracy and robustness of the model were evaluated on an unseen test set after training was guided by performance measures on a validation set. The model architecture, training procedure, and ultimate performance metrics are explained in detail in the subsequent sections.

4.2.1 Model Architecture

This study's classification model is a deep learning architecture built on Long Short-Term Memory (LSTM) networks, which are very good at identifying temporal dynamics and sequential dependencies in data. The `Keras` API was used to create the model, which has a sequential architecture made up of several layers intended to gradually extract and analyze temporal patterns.

The architecture begins with an LSTM layer consisting of 128 units, configured with `return_sequences=False` to allow the output to be passed as a sequence to the subsequent LSTM layer. Immediately following this, a dropout layer is applied at a rate of 0.2 to prevent overfitting by randomly deactivating some of the neurons during training. A second 64-unit LSTM layer comes next, where `return_sequences=False` makes sure that just the sequence's final output is sent on. Here, a second dropout layer is added to preserve regularization.

After the recurrent layers, a fully connected (`Dense`) layer with 64 neurons and ReLU activation is used to introduce non-linearity and compress the representation. The output layer, a dense layer with a number of units equal to the number of target classes (15), is applied after a final dropout layer. The output layer generates a probability distribution across the classes that is suitable for multiclass classification using the softmax activation function.

The input to the model is represented as a three-dimensional tensor

$$X \in \mathbb{R}^{(n_{\text{samples}},\, n_{\text{timesteps}},\, n_{\text{features}})},$$

commonly denoted as (s, t, f), where:

- $s = n_{\text{samples}}$: number of samples (batch size),
- $t = n_{\text{timesteps}}$: number of timesteps,
- $f = n_{\text{features}}$: number of features per timestep.

Each sample in X can be viewed as a matrix of shape $(t \times f)$, representing the temporal evolution of feature values over time. Formally, the full tensor can be expressed as:

$$X = \left\{ \begin{bmatrix} x_{1,1}^{(1)} & x_{1,2}^{(1)} & \cdots & x_{1,f}^{(1)} \\ x_{2,1}^{(1)} & x_{2,2}^{(1)} & \cdots & x_{2,f}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_{t,1}^{(1)} & x_{t,2}^{(1)} & \cdots & x_{t,f}^{(1)} \end{bmatrix}, \quad \ldots, \quad \begin{bmatrix} x_{1,1}^{(s)} & x_{1,2}^{(s)} & \cdots & x_{1,f}^{(s)} \\ x_{2,1}^{(s)} & x_{2,2}^{(s)} & \cdots & x_{2,f}^{(s)} \\ \vdots & \vdots & \ddots & \vdots \\ x_{t,1}^{(s)} & x_{t,2}^{(s)} & \cdots & x_{t,f}^{(s)} \end{bmatrix} \right\},$$

where $x_{i,j}^{(k)}$ denotes the value of feature j at timestep i for sample k.

In this work, the dataset was reshaped such that $n_{\text{features}} = 1$ using `np.expand_dims`, resulting in an input shape of $(n_{\text{samples}}, n_{\text{timesteps}}, 1)$. This structure is required by LSTM layers, which process one feature vector at a time across the defined temporal sequence.

A graphical representation of the network architecture is provided in Figure 4.8, while Table 4.1 presents a summary of its configuration.

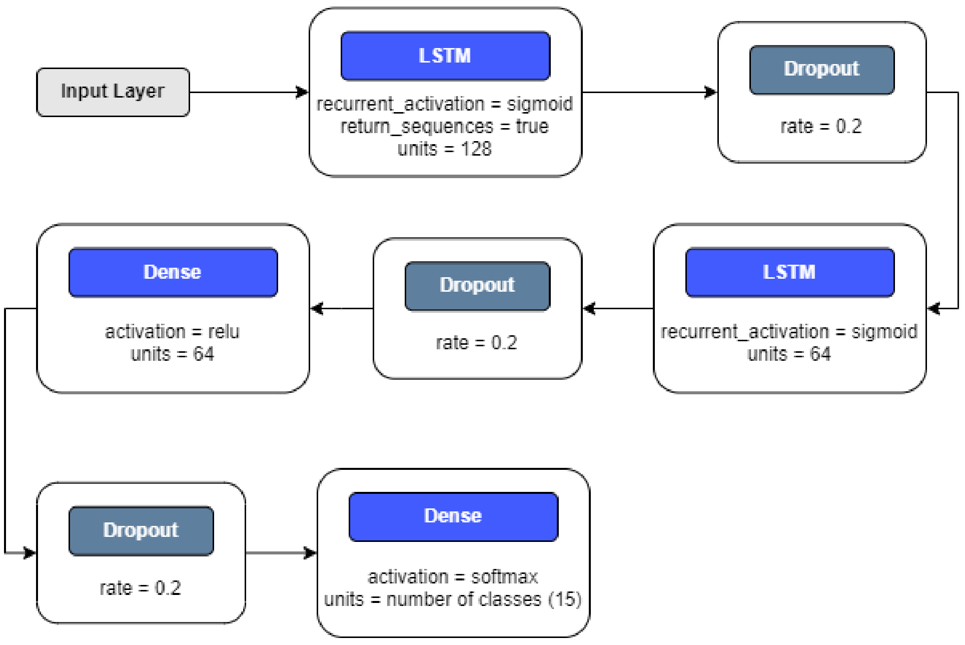


Figure 4.8: Model Architecture.

Layer Type	**Configuration**
Input Layer	Shape: (features, 1) after preprocessing and reshaping
LSTM Layer 1	128 units, return_sequences=True, recurrent_activation=sigmoid
Dropout Layer	Rate: 0.2
LSTM Layer 2	64 units, return_sequences=False, recurrent_activation=sigmoid
Dropout Layer	Rate: 0.2
Dense Layer	64 units, ReLU activation
Dropout Layer	Rate: 0.2
Output Layer	15 units, Softmax activation

Table 4.1: Summary of the LSTM model architecture.

4.2.2 Compilation and Training

Following the construction of the model architecture, the network was compiled and trained using supervised learning. The optimizer, loss function, and evaluation metrics have to be specified at the compilation process. The categorical cross-entropy loss function was used because the challenge involved multiclass categorization. When the target labels are one-hot encoded, as was done during the preprocessing phase, this loss function is suitable.

The Adam optimizer with a learning rate of 0.001 was used to optimize the model.

Adam was chosen because of its effectiveness and capacity for adaptable learning rates, which make it a good fit for high-dimensional, complex optimization issues like those that deep neural networks present.

Input Reshaping for LSTM

LSTM models require input in the form of a 3D tensor. In this study, each input sample consists of a fixed number of features (after all preprocessing steps), but not multiple timesteps. Therefore, the data was reshaped by increasing the feature dimension so that each feature vector is viewed as a "sequence" of one timestep and several features in order to adjust the feature matrix to the predicted input shape of the LSTM. The `expand_dims` function in NumPy was used for this, adding a singleton dimension at the end:

```
train_features_reshaped = np.expand_dims(train_features, axis=-1)
```

This reshaping allows the LSTM to process each sample as a sequence of scalar features (over 1 time step), enabling compatibility with the recurrent layer input specification.

Training Strategy

Accuracy was the main metric used to assess the model's performance. Early stopping was used during training to enhance generalization and avoid overfitting. If no improvement was seen for five consecutive epochs, the early stopping mechanism stopped training after monitoring the validation loss. Additionally, the best-performing weights on the validation set were restored at the end of training.

The model was trained for a maximum of 50 epochs with a batch size of 32. Mini-batches of training data were fed into the model, and a distinct validation subset selected from the original dataset was used to validate the model at the conclusion of each epoch. Accuracy and loss curves from the training history were captured and subsequently displayed to evaluate convergence and identify any indications of overfitting.

Training was conducted on the oversampled training set (SMOTE-applied) and evaluated after each epoch on the validation set, which was also oversampled using the same technique to maintain class balance.

Performance Monitoring

As said previously, training and validation performance was monitored using accuracy and loss metrics. These were recorded throughout the training process and visualized to evaluate convergence behavior and detect overfitting. The resulting plots are shown in Figures 4.9 and 4.10.

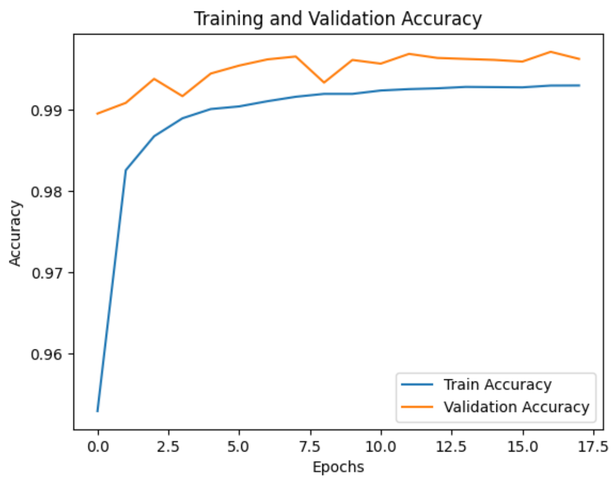


Figure 4.9: Training and validation accuracy across epochs.

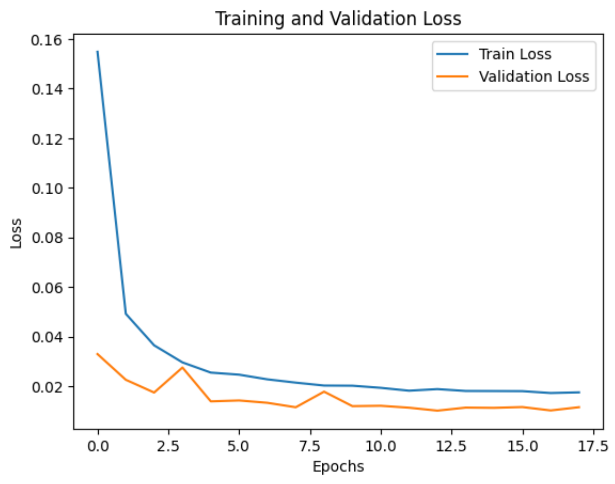


Figure 4.10: Training and validation loss across epochs.

4.2.3 Evaluation on Test Data

After the training phase, the final model was evaluated on the held-out test set, which was not used during training or validation. This evaluation aimed to assess the model's generalization capability and its performance in classifying previously unseen data.

Preparation and Prediction

As in training and validation, the test feature matrix was reshaped prior to inference to conform to the LSTM input format.

Class probabilities were then produced for every test occurrence using the trained model. By selecting the index of the maximum probability (`argmax`) corresponding to the predicted class, these probabilities were transformed into discrete class predictions.

Quantitative Evaluation Metrics

To quantify the model's performance, several standard classification metrics were computed:

- **Accuracy**: The overall proportion of correct predictions.
- **Precision, Recall, and F1-Score**: Computed per class, these metrics provide insights into how well the model identifies each class, especially important in imbalanced datasets.
- **Confusion Matrix**: A visual summary of true versus predicted class distributions, highlighting the types of misclassifications.

The following subsections present the evaluation results in detail, including the confusion matrices and per-class performance metrics.

Confusion Matrix

The confusion matrices presented in Figures 4.11 and 4.12 provide a detailed visualization of the models classification performance. Each matrix compares the predicted and actual classes, helping to identify where the model performs well and where misclassifications occur.

Figure 4.11 shows the raw count of predictions for each class, where diagonal elements represent correct classifications and off-diagonal elements correspond to misclassifications. In contrast, Figure 4.12 displays the same results normalized by the number of true instances per class, offering a clearer comparison across classes with different sample sizes. This normalization highlights relative performance, particularly revealing that minority classes such as *Infiltration*, *XSS*, and *SQL Injection* exhibit lower recall compared to the majority categories.

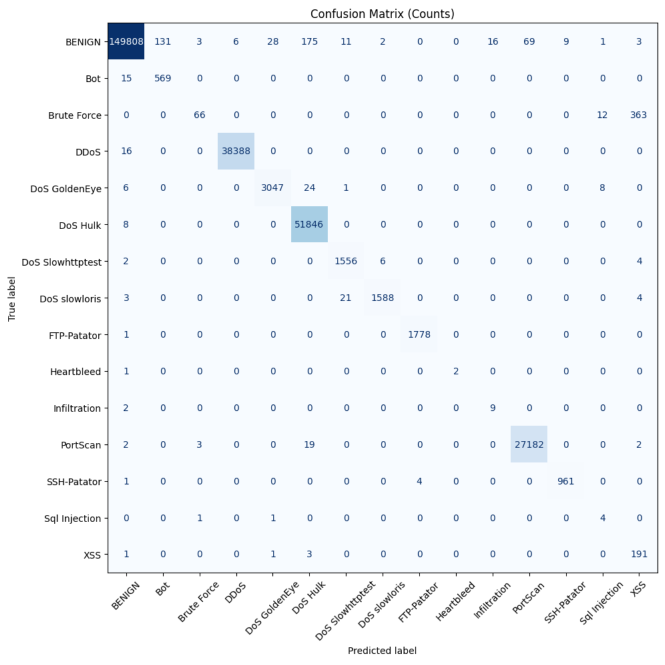


Figure 4.11: Confusion matrix of model predictions on the test set.

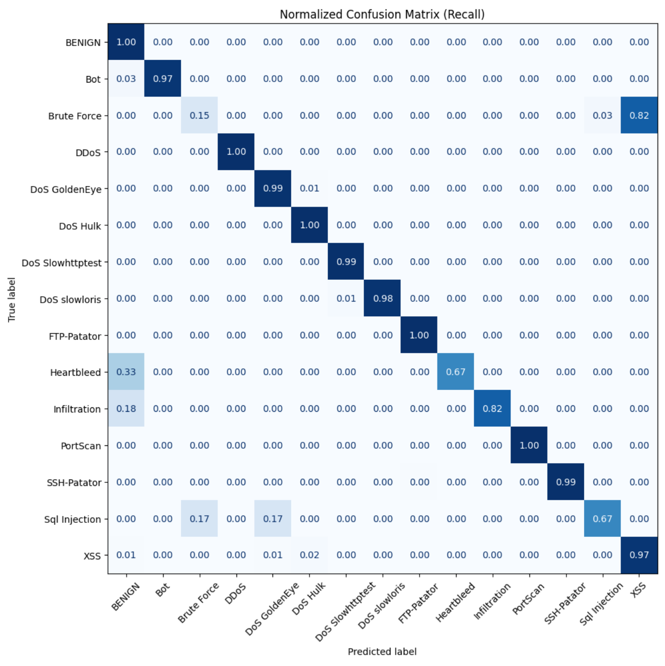


Figure 4.12: Normalized confusion matrix of model predictions on the test set.

Classification Report

The classification report presented in Table 4.2 includes precision, recall, and F1-score for each class.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
BENIGN	0.9995	0.9973	0.9984	150262
Bot	0.8117	0.9743	0.8856	584
Brute Force	0.9398	0.1769	0.2977	441
DDoS	0.9999	0.9997	0.9998	38404
DoS GoldenEye	0.9954	0.9854	0.9904	3086
DoS Hulk	0.9961	0.9995	0.9978	51854
DoS Slowhttptest	0.9811	0.9917	0.9864	1568
DoS slowloris	0.9956	0.9833	0.9894	1616
FTP-Patator	0.9961	0.9978	0.9969	1779
Heartbleed	1.0000	0.6667	0.8000	3
Infiltration	0.4737	0.8182	0.6000	11
PortScan	0.9975	0.9991	0.9983	27208
SSH-Patator	1.0000	0.9917	0.9958	966
Sql Injection	0.2105	0.6667	0.3200	6
XSS	0.3345	0.9898	0.5000	196
Accuracy			0.9966	277984
Macro Avg	0.8488	0.8825	0.8238	277984
Weighted Avg	0.9976	0.9966	0.9965	277984

Table 4.2: Classification report showing precision, recall, and F1-score per class on the test set.

Overall Accuracy

The model achieved a test accuracy of 99.64% on the held-out test set (277,984 samples), computed using the `accuracy_score` function from the scikit-learn library. While this high accuracy suggests strong overall performance, it is influenced by the dominance of certain classes, which can mask weaker predictions on minority categories. To account for this, macro and weighted-averaged metrics were evaluated. The macro metrics, which treat all classes equally, yielded a precision of 0.83, recall of 0.88, and F1-score of 0.81, highlighting the challenges in correctly identifying rare attacks. In contrast, the weighted metrics, accounting for class frequency, remained close to the overall accuracy (precision: 0.997, recall: 0.996, F1-score: 0.996).

The per-class classification report and confusion matrices (Figures 4.11 and 4.12) provide further insight into these discrepancies. While majority classes such as BENIGN, DDoS, DoS Hulk, and PortScan achieved near-perfect detection, minority classes like Brute Force, Infiltration, SQL Injection, and XSS showed considerably lower recall or precision. These results emphasize that, despite strong aggregate performance, the model still struggles with certain low-frequency attack types, underscoring the importance of evaluating beyond accuracy when dealing with imbalanced datasets.

4.2.4 Model Saving and Re-usability

TensorFlow's built-in `model.save()` method was used to store the trained LSTM model in the `.keras` format for easier use and deployment in the future. The model architecture and learnt weights are both preserved in this format, allowing for easy reloading for assessment, inference, or more training. Reloading the saved model with `tensorflow.keras.models.load_model()` ensures complete reproducibility and portability of the trained solution without needing reconstruction of the original architecture.

Chapter 5

System Architecture

This chapter presents the architecture of the proof-of-concept system developed for network attack prediction. The goal of the system is to demonstrate the feasibility of using deep learning techniques to identify and classify malicious network activity based on structured network traffic data.

The architecture is designed to handle the end-to-end pipeline required for this task, starting from raw data ingestion, through preprocessing and feature extraction, to model inference and result reporting and storage. Each component of the system is modular, allowing for flexibility in experimentation and future expansion.

The design prioritizes clarity, reproducibility, and integration with commonly used data science tools and libraries. The components are implemented using a combination of open-source technologies to ensure accessibility and ease of deployment.

A high-level diagram of the system architecture is shown in Figure 5.1 to illustrate the main components and the flow of data through the system. The architecture is composed of modular elements distributed across different machines that cooperate to enable near real-time attack prediction. Network traffic is first captured and processed on the IDS/IPS/Victim machine using the Flow Extractor (CICFlowMeter) and stored in the file system in a CSV file. The Flow Shipper module then sends the extracted flow records through a Redis queue to the DeepShield server, which performs deep learning inference using the trained LSTM model. Identified malicious connections are subsequently sent back and stored for visualization and analysis in the InfluxDB database. The following sections describe each component in detail and its role in achieving accurate and efficient network attack prediction.

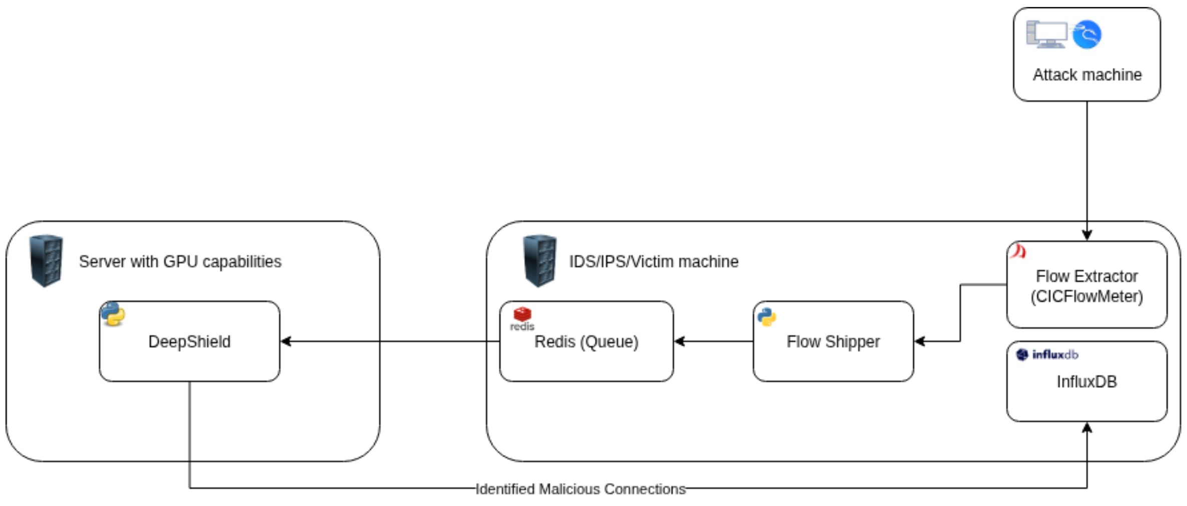


Figure 5.1: High-level system architecture for network attack prediction.

5.1 System Objective

The primary objective of the proposed system is to detect malicious network activity by analyzing captured network traffic and classifying each connection as either benign or belonging to a specific type of attack. This is achieved through a pipeline that transforms raw packet data into a machine learning-compatible format, enabling near real-time prediction using a deep learning model.

To support this objective, the system is architected around three main components, distributed across three separate machines:

- The **Attack Machine**, which acts as a real malicious actor machine, making real network attacks to the victim machine.
- The **IDS/IPS/Victim Machine**, which acts as both a data source and preprocessor. It captures network traffic, extracts flow-based features using **CICFlowMeter**. These flows are then shipped via a **Redis** queue for further analysis and received again with a prediction (if it is malicious) and stored in an **InfluxDB**.
- The Model Machine, **DeepShield**, detailed in Sections 5.2.5 and 5.3.1, is a high-performance server equipped with GPU capabilities, which runs the DL model. This machine consumes the flow records from the **Redis** queue and performs inference to determine whether a connection is malicious. If a malicious connection is identified, the system takes further action, sending the alert to a **InfluxDB**.

This distributed architecture allows the data collection and model inference processes to scale independently and supports modular experimentation with different flow extraction, queuing, and modeling strategies.

5.2 Architecture Components

The architecture of the network attack prediction system consists of several modular components, each dedicated to a specific stage within the detection pipeline. These components are primarily distributed across two main machines: the **IDS/IPS/Victim Machine** and the **Model Machine** (DeepShield Machine), as previously introduced. In addition, a third machine, the **Attack Machine**, is included for the purpose of generating and simulating malicious traffic to test and validate the system's effectiveness. The following subsections describe each component in detail, outlining its functionality and role within the overall architecture.

5.2.1 Attack Machine

The attack machine, running **Kali Linux**, simulates malicious behavior within the network by launching a variety of targeted attacks against the victim machine. Although it operates outside the main detection pipeline, it plays a critical role in generating realistic traffic. This controlled setup enables thorough testing and validation of the systems detection capabilities.

5.2.2 Flow Extractor (CICFlowMeter)

In the **Victim Machine**, raw network traffic is initially captured as Packet Capture (PCAP) files. These captures are then processed using **CICFlowMeter**, an open-source tool that transforms packet-level data into bidirectional flow-based records. In this setup, a **Python** utility is used to capture packets from a specified network interface and process them in real-time. Each resulting flow contains statistical features such as duration, packet and byte counts, and others, essential inputs for this machine learning analysis. The resulting flows are stored in a CSV file.

5.2.3 Flow Shipper

The **Flow Shipper** is a Python-based module that reads flows from a CSV file, and publishes them to a **Redis** queue. This module ensures that the CSV file lines are preprocessed in a format that when forwarded to the model server its processing is easy, optimizing performance and reducing overhead.

5.2.4 Redis (Queue)

Redis acts as a lightweight, in-memory queue for asynchronous communication between the victim machine and the model machine. It ensures decoupling of data ingestion and model inference stages, allowing the system to handle varying data rates and processing loads. Redis is well-suited for this task due to its speed, reliability, and pub-sub capabilities.

5.2.5 DeepShield (Model Inference)

Running on the **Model Machine**, **DeepShield** functions as the primary prediction module leveraging deep learning techniques. It consumes flow records from a **Redis** queue, applies preprocessing steps, including feature selection and transformation using the same **Quantile Transformer** employed during training, and performs inference via a GPU-accelerated neural network. Each network packet is classified as either benign or as a specific type of network attack, with malicious classifications stored in an **InfluxDB** instance for further analysis. The use of GPU resources ensures high-throughput, low-latency predictions, making the system suitable for real-time or near-real-time network threat detection.

Identified Malicious Connections

When the model classifies a flow as malicious, the alert is sent back to the victim machine through a direct HTTP request to the InfluxDB instance. Using the official InfluxDB Python client, the system writes an `alert` measurement containing the predicted class, message ID, full flow record, and timestamp. This write operation uses the InfluxDB API with an authentication token, ensuring secure communication. Once the alert is stored, the corresponding Redis entry is acknowledged and deleted, completing the feedback loop between the inference module and the database.

5.2.6 InfluxDB

Extracted flows are stored in an **InfluxDB** database, providing a persistent and flexible schema for querying and organizing the flow data. **InfluxDB** serves as an intermediate storage layer, allowing the system to decouple flow extraction from real-time inference. This also facilitates auditing and debugging.

5.3 Technology Stack

The system is built using a collection of open-source tools and frameworks that support modularity, scalability, and performance. The selected technologies facilitate efficient handling of large volumes of network traffic data and support GPU-accelerated machine learning inference.

Component	Technology	Purpose
Flow Extraction	CICFlowMeter	Converts raw PCAPs to flow-based feature vectors
Database	InfluxDB	Stores extracted flow records
Queueing	Redis	Lightweight message queue between modules
Flow Shipping	Python (csv, redis-py)	Retrieves data from a CSV file and publishes it to Redis
Model Inference	DeepShield (Python, TensorFlow, pandas, numpy)	Deep learning-based attack prediction
Infrastructure	Linux (Ubuntu)	Operating system for all machines
GPU Acceleration	NVIDIA CUDA-enabled GPUs	Enables fast model inference
Visualization/Logging	Log outputs	For output inspection and auditing

Table 5.1: System components, technologies used, and their respective purposes.

5.3.1 Model Inference Machine Hardware Specifications

The hardware specifications are provided exclusively for the Model Machine, as it performs the most computationally intensive tasks within the system, namely deep learning inference. Accurate reporting of its configuration is essential for understanding the system's performance characteristics and for enabling reproducibility in future research. In contrast, the IDS/IPS/Victim Machine and the Attack Machine perform less demanding operations such as packet capture, or traffic simulation, all of which can be executed on standard consumer-grade hardware without significantly impacting system performance.

The Model Machine used for running the DeepShield inference engine is equipped with the following hardware:

- **CPU**: 2 Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz
- **GPU**: 2 Nvidia TITAN V 12GiB
- **RAM**: 94GB
- **Storage**: 465.8GB
- **OS**: Ubuntu 22.04.4 LTS

5.4 Deployment Considerations

The system has been designed with deployment flexibility and maintainability in mind, leveraging containerization through **Docker Compose** on both the

IDS/IPS/Victim Machine and the **Model Machine**. Each major component, such as **CICFlowMeter**, the **InfluxDB** database, the **Flow Shipper**, the **Redis** queue, and the **DeepShield** inference engine is encapsulated in its own **Docker** container. This modular container-based approach simplifies installation, environment configuration, and dependency management across different systems.

Using Docker Compose enables all components on a given machine to be orchestrated as a single multi-container application. This greatly reduces the manual overhead of setup and ensures consistency between development, testing, and production environments. Furthermore, this structure allows for straightforward scaling or substitution of individual services. For example, the deep learning model container can be updated or replaced independently of the data pipeline components.

The decoupled nature of the system, along with Docker-based deployment, makes it suitable for a range of use cases, from lab-based testing to experimental real-world setups, and supports both horizontal scaling and integration with cloud-native orchestration platforms in future work.

5.5 Summary

This chapter described the architecture of the proposed system for network attack prediction, emphasizing its modular and distributed design. The system spans two main machines: one dedicated to network traffic collection, and another optimized for flow processing and deep learning inference. An additional attack machine is used to simulate malicious traffic for evaluation purposes.

A key feature of the implementation is the use of Docker Compose to deploy system components as containers on both the victim and model machines. This containerized approach enhances reproducibility, simplifies system setup, and allows individual services to be developed, tested, and deployed in isolation. The architecture is further reinforced by asynchronous communication via Redis, GPU-accelerated inference, and persistent storage using InfluxDB.

The design choices made in this system support not only the current proof-of-concept implementation but also future scalability and integration with more advanced security infrastructures. The following chapter evaluates the performance and accuracy of the system under real and simulated attack conditions.

Chapter 6

Experimental Evaluation and Results

This chapter presents the testing and evaluation of the proposed intrusion detection system. The goal of testing is twofold:

- Verify that the system runs correctly end-to-end, from traffic capture to alert visualization.
- Evaluation of the systems ability to detect malicious activity in real network conditions.

The system was tested by generating controlled attacks from an attacker machine against a designated victim machine, with traffic being processed by the detection pipeline and alerts visualized through InfluxDB dashboards.

Due to resource constraints and scope limitations, only a subset of possible attacks was tested. Some attack scenarios (such as web-based exploits, SQL injection, or cross-site scripting) would have required setting up additional infrastructure such as vulnerable web servers or specific application environments, which falls outside the scope of this dissertation. Instead, testing focused on network-level attacks that could be executed within the available environment, such as DoS Slowhttptest, DoS Slowloris, Port Scans, and SSH Patator.

6.1 Testing Methodology

6.1.1 Environment Setup

Testing was done with the same system presented in Chapter 5. All components were deployed in a controlled lab network.

6.1.2 Attack Scenarios

To validate detection capabilities, a range of attacks were executed from the attacker machine against the victim machine. These included DoS Slowhttptest (Section 6.2.1), DoS Slowloris (Section 6.2.2), Port Scans (Section 6.2.3), and SSH Patator (Section 6.2.4), as mentioned in the introduction of the chapter.

6.1.3 Data Collection and Annotation

- The attacks start and stop times were manually annotated.
- Flow-level features were extracted in near-real time.
- The predictions of the system were compared to the ground truth (attack timestamps).

6.1.4 Evaluation Metrics

The following evaluation criteria were used:

- **Detection Success**: whether the system generated alerts corresponding to the attack window.
- **Volume of Alerts**: number of flows flagged as malicious.
- **Timeliness**: whether alerts appeared during or shortly after the execution of the attack.
- **False Positives**: alerts outside the annotated attack interval.

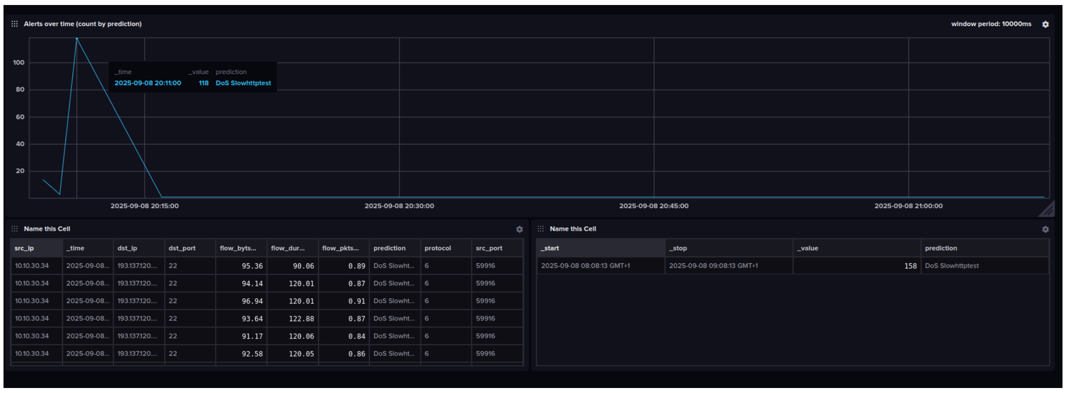
6.2 Detection Examples

6.2.1 DoS Slowhttptest

Slowhttptest generates a Denial-of-Service condition by opening multiple connections to the victim and sending partial requests at a very slow rate. This keeps server resources tied, preventing normal clients from being served.

Observed Alerts

During the experiment, which ran from 20:00 to 21:10, the system correctly classified the malicious activity as **DoS Slowhttptest**. As illustrated in Figure 6.1, a total of **158 connections (flows)** were identified as malicious throughout the attack period, with a noticeable peak of **118 detections** occurring at 20:11.



_time	_value	prediction
2025-09-08 20:11:00	118	DoS Slowhttptest

src_ip	_time	dst_ip	dst_port	flow_byts...	flow_dur...	flow_pkts...	prediction	protocol	src_port
10.10.30.34	2025-09-08...	193.137.120....	22	95.36	90.06	0.89	DoS Slowht...	6	59916
10.10.30.34	2025-09-08...	193.137.120....	22	94.14	120.01	0.87	DoS Slowht...	6	59916
10.10.30.34	2025-09-08...	193.137.120....	22	96.94	120.01	0.91	DoS Slowht...	6	59916
10.10.30.34	2025-09-08...	193.137.120....	22	93.64	122.88	0.87	DoS Slowht...	6	59916
10.10.30.34	2025-09-08...	193.137.120....	22	91.17	120.06	0.84	DoS Slowht...	6	59916
10.10.30.34	2025-09-08...	193.137.120....	22	92.58	120.05	0.86	DoS Slowht...	6	59916

_start	_stop	_value	prediction
2025-09-08 08:08:13 GMT+1	2025-09-08 09:08:13 GMT+1	158	DoS Slowhttptest

Figure 6.1: DoS Slowhttptest attack experiment detected alerts

In this experiment, the tool **pentmenu** was used.

Discussion

The system effectively detected the attack in near real time, maintaining consistent alert generation throughout its duration. This result highlights the models capability to identify DoS traffic patterns that resemble those encountered during the training phase.

6.2.2 DoS Slowloris

Slowloris sends HTTP headers very slowly, maintaining many simultaneous connections and gradually exhausting the target servers resources without requiring high bandwidth.

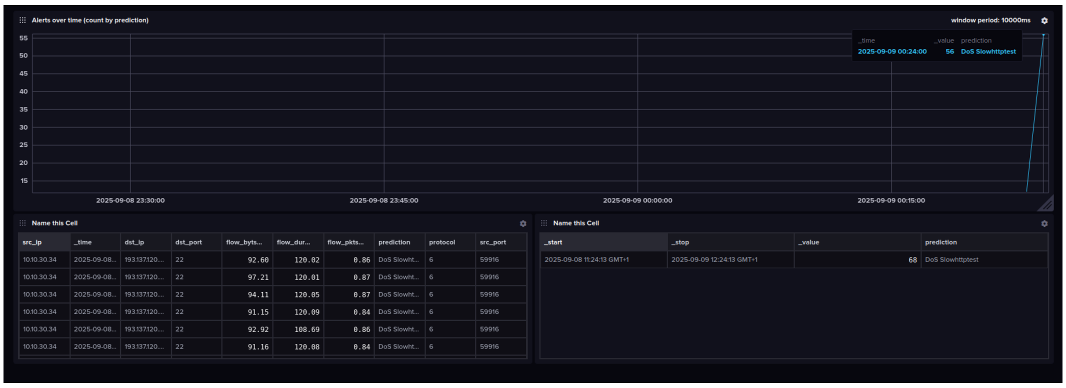
Observed Alerts

The experiment, which ran from 00:10 to 00:30, produced multiple alerts identified as **DoS Slowhttptest** during the attack. As shown in Figure 6.2, a total of **68 connections (flows)** were detected as malicious throughout the attack period, with a peak of **56 detections** occurring at 00:24.

To carry out this experiment, the same tool **pentmenu** was used.

Discussion

The system successfully detected the presence of an attack but misclassified its specific type, identifying a similar variant instead. This outcome highlights the models limitation in distinguishing between slow DoS attack types, namely **Slowhttptest** and **Slowloris**.



src_ip	_time	dst_ip	dst_port	flow_byts...	flow_dur...	flow_pkts...	prediction	protocol	src_port
10.10.30.34	2025-09-08...	193.137.120....	22	92.60	120.02	0.86	DoS Slowht...	6	59916
10.10.30.34	2025-09-08...	193.137.120....	22	97.21	120.01	0.87	DoS Slowht...	6	59916
10.10.30.34	2025-09-08...	193.137.120....	22	94.11	120.05	0.87	DoS Slowht...	6	59916
10.10.30.34	2025-09-08...	193.137.120....	22	91.15	120.09	0.84	DoS Slowht...	6	59916
10.10.30.34	2025-09-08...	193.137.120....	22	92.92	108.69	0.86	DoS Slowht...	6	59916
10.10.30.34	2025-09-08...	193.137.120....	22	91.16	120.08	0.84	DoS Slowht...	6	59916

_start	_stop	_value	prediction
2025-09-08 11:24:13 GMT+1	2025-09-09 12:24:13 GMT+1	68	DoS Slowhttptest

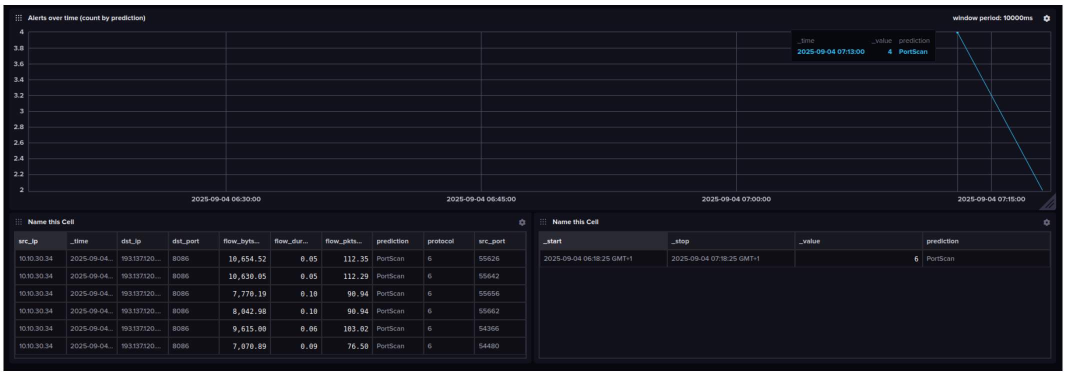
Figure 6.2: DoS Slowloris attack experiment detected alerts

6.2.3 Port Scans

To test reconnaissance detection, the **Nmap** tool was used to perform a port scan on the victim machine. **Nmap** systematically probes a range of ports to identify open services.

Observed Alerts

The system generated alerts labeled **PortScan**, with a total of **6 connections (flows)** identified as malicious throughout the scan period and a peak of **4 detections** observed at 6:57. These alerts were concentrated during the scan activity, which lasted from 6:57 to 7:18.



src_ip	_time	dst_ip	dst_port	flow_byts...	flow_dur...	flow_pkts...	prediction	protocol	src_port
10.10.30.34	2025-09-04...	193.137.120....	8086	10,654.52	0.05	112.35	PortScan	6	55626
10.10.30.34	2025-09-04...	193.137.120....	8086	10,630.05	0.05	112.29	PortScan	6	55642
10.10.30.34	2025-09-04...	193.137.120....	8086	7,770.19	0.10	90.94	PortScan	6	55656
10.10.30.34	2025-09-04...	193.137.120....	8086	8,042.98	0.10	90.94	PortScan	6	55662
10.10.30.34	2025-09-04...	193.137.120....	8086	9,615.00	0.06	103.02	PortScan	6	54366
10.10.30.34	2025-09-04...	193.137.120....	8086	7,070.89	0.09	76.50	PortScan	6	54480

_start	_stop	_value	prediction
2025-09-04 06:18:25 GMT+1	2025-09-04 07:18:25 GMT+1	6	PortScan

Figure 6.3: PortScan attack experiment detected alerts

Discussion

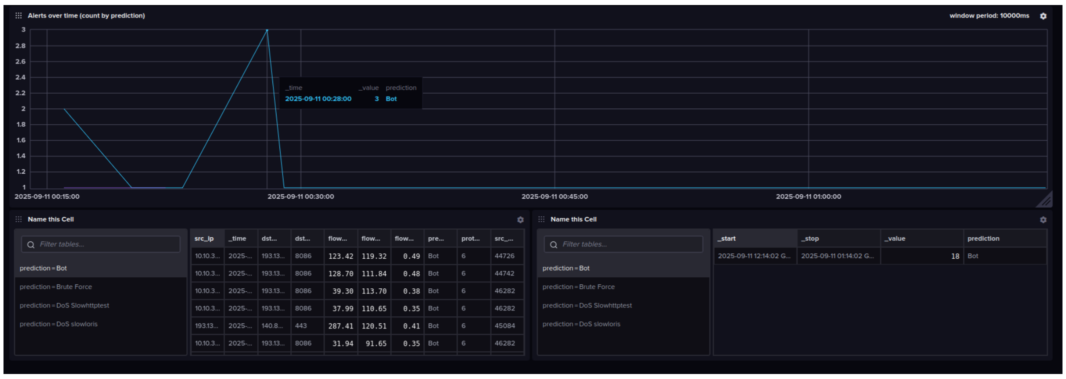
Detection was immediate and accurate, demonstrating the systems strong capability to recognize scanning behavior. The results confirm that it can effectively identify reconnaissance activity, which often serves as a precursor to more complex intrusions. However, the system shows a limitation in generating a high number of alerts when the victim system has only a few active ports.

6.2.4 SSH Patator (Brute Force)

The **Patator** tool was used to execute a brute force attack against the victims SSH service (port 22). **Patator** attempts multiple login combinations in quick succession, aiming to guess valid credentials and gain unauthorized access.

Observed Alerts

The experiment, conducted between 00:15 and 01:30, showed that the system generated alerts during the attack window but incorrectly labeled them as **Bot** instead of **SSH-Patator**. As shown in Figure 6.4, the **InfluxDB** dashboard recorded a total of **18 alerts** throughout the attack period, with a peak of **3 detections** occurring at 00:28.



_time	_value	prediction
2025-09-11 00:28:00	3	Bot

src_ip	_time	dst...	dst...	flow...	flow...	flow...	pre...	prot...	src_...
10.10.3...	2025-...	193.13...	8086	123.42	119.32	0.49	Bot	6	44726
10.10.3...	2025-...	193.13...	8086	128.70	111.84	0.48	Bot	6	44742
10.10.3...	2025-...	193.13...	8086	39.30	113.70	0.38	Bot	6	46282
10.10.3...	2025-...	193.13...	8086	37.99	110.65	0.35	Bot	6	46282
193.13...	2025-...	140.8...	443	287.41	120.51	0.41	Bot	6	45084
10.10.3...	2025-...	193.13...	8086	31.94	91.65	0.35	Bot	6	46282

_start	_stop	_value	prediction
2025-09-11 12:14:02 G...	2025-09-11 01:14:02 G...	18	Bot

Figure 6.4: SSH Patator attack experiment detected alerts

Discussion

The system clearly detected abnormal traffic during the brute force attack execution. However, instead of correctly classifying the attack as SSH-Patator, it was mislabeled as Bot. This misclassification highlights one of the systems generalization limitations: while it can recognize that malicious activity is taking place, it may not always assign the correct attack label when encountering tools or behaviors that differ slightly from those in the training data. Nevertheless, for practical purposes, the system still generated valuable alerts, allowing security operators to investigate the suspicious activity regardless of the precise classification.

6.3 Summary

This chapter demonstrated the testing and evaluation of the proposed intrusion detection system using controlled attack scenarios. Due to resource constraints and scope limitations, only a subset of attacks was executed, focusing on those that could be reproduced in the available environment: **DoS Slowhttptest**, **DoS Slowloris**, **Port Scans (Nmap)**, and **SSH Patator**.

The results confirm that the system is **functional end-to-end**, successfully capturing network traffic, generating flow-level features, running real-time inference, and presenting alerts through **InfluxDB** dashboards. Across the tested scenarios, the system consistently detected abnormal behavior:

- **DoS Slowhttptest** were correctly classified with a high number of alerts, confirming robustness against this type of low-rate denial-of-service.
- **DoS Slowloris** was detected but but misclassified as *DoS Slowhttptest*, showing the system's difficulty in differentiate different types of low-rate denial-of-service.
- **Port Scans** were detected immediately and accurately, showing the systems ability to identify reconnaissance activity.
- **SSH Patator** was detected but misclassified as *Bot*, illustrating that while the system recognizes malicious patterns, it does not always assign the exact correct attack label.

Overall, the evaluation highlights two key findings:

1. **Strengths** The system reliably detects malicious network activity in real time, proving that the architecture and methodology are valid.
2. **Limitations** Misclassifications and incomplete coverage for new or application-specific attacks show the generalization gap of supervised machine learning models trained on predefined datasets.

These findings align with common challenges in intrusion detection research and reinforce the need for future improvements, such as retraining with more diverse datasets, integrating anomaly detection techniques, or adopting incremental learning approaches.

Despite its limitations, the system fulfills its primary goal: to provide a proof-of-concept demonstration of machine learning-based intrusion detection in a real-world deployment setting.

Chapter 7

Conclusion and Future Work

This dissertation examined the feasibility and effectiveness of deep learning for network attack prediction, with emphasis on Long Short-Term Memory (LSTM) networks trained on CIC-IDS2017. Motivated by the escalating complexity and velocity of cyber threats and the limitations of static, signature-based approaches, the work advanced both a learning pipeline and a deployable proof-of-concept architecture for near real-time intrusion prediction.

7.1 Summary of Contributions

The main contributions are as follows:

A data engineering pipeline was developed to handle the CIC-IDS2017 dataset, encompassing feature reduction, cleaning, balancing, and transformation. This process included correlation-driven pruning and quantile-based scaling, resulting in compact yet highly discriminative inputs suitable for sequential modeling.

Building upon this foundation, an LSTM-based multiclass classifier was designed and trained, achieving a held-out test accuracy of **99.64%**. Beyond accuracy, additional metrics demonstrated robust performance, with a macro precision of **0.8346**, macro recall of **0.8801**, macro F1-score of **0.8100**, and weighted precision of **0.9974**.

To validate its practical applicability, a modular and containerized system architecture was implemented, integrating flow extraction, queuing, inference, and alerting components through tools such as CICFlowMeter, Redis, and InfluxDB. This architecture demonstrated the feasibility of near real-time operation and provided a flexible framework for experimental evaluation.

7.2 Empirical Findings

The trained model exhibited high overall discriminative ability on the benchmark dataset, with near-perfect detection for prevalent classes (e.g., BENIGN, DDoS, DoS Hulk, PortScan). Controlled experiments further validated end-to-end functionality:

- *DoS Slowhttptest*: consistently detected and correctly labeled during the attack window.
- *DoS Slowloris*: reliably detected but conflated with Slowhttptest, indicating difficulty in fine-grained discrimination among closely related low-rate DoS variants.
- *Port scans*: promptly and accurately identified, evidencing robustness for reconnaissance activity.
- *SSH Patator*: abnormal activity was detected though mislabeled (e.g., as Bot), reflecting the challenges of class granularity under distributional changes.

These results collectively substantiate the central hypothesis: deep sequence models trained on representative flow features can deliver high multiclass prediction performance and can be operationalized within a practical architecture.

7.3 Discussion

The obtained results highlight several important points.

First, relying only on overall accuracy can hide how well the model performs on less frequent attack types. Metrics such as macro-averaged precision, recall, and F1-score, together with the confusion matrix, were essential to understand the behavior of these minority but security-critical classes.

Second, some attacks that are very similar in nature, such as low-rate DoS variants, were occasionally confused with one another. This suggests that the predefined categories in benchmark datasets like CIC-IDS2017 do not always reflect real operational differences.

Finally, when compared to other deep learning architectures presented in Section 3.2, the LSTM approach achieved competitive or superior results on sequential network flow data. This confirms that LSTM networks are well suited for temporal traffic analysis, while other architectures might still offer advantages in feature extraction or scalability depending on the scenario.

7.4 Limitations and Threats to Validity

The system presents several limitations that should be acknowledged. First, its strong dependence on the CIC-IDS2017 dataset constrains representativeness, as changes in modern network traffic patterns and attack tools can reduce the external validity of the model. Despite mitigation strategies such as SMOTE, class imbalance remains a challenge, particularly for minority attack types, which affects macro-level performance metrics and contributes to occasional misclassifications.

Moreover, the experimental scope focused primarily on a subset of network-layer attacks, excluding application-layer exploits that would require dedicated infrastructure for realistic testing. Finally, although near real-time detection was successfully demonstrated, comprehensive benchmarking of end-to-end throughput and latency under varying load conditions was not conducted, limiting the ability to fully characterize the systems performance boundaries.

7.5 Future Work

Several avenues can extend this research:

1. **Data diversification and realism:** The usage of newer and more varied datasets collected from different network sources and over longer periods. This would help test how well the model adapts to changes in real network environments.

2. **Learning under imbalance:** Exploring methods such as cost-sensitive training (giving more importance to rare attacks), adjusting classification thresholds, or using smarter sampling strategies to improve detection of less frequent attack types.

3. **Architectural innovation:** Experimentation with newer deep learning models like Temporal CNNs or Transformers, as well as hybrid or semi-supervised approaches that can learn from both labeled and unlabeled data.

4. **Robustness and adaptability:** Testing how the model performs against adversarial attacks and changing traffic patterns, and explore continuous or online learning methods to keep the model updated over time.

5. **System optimization:** Performance measure in terms of latency and throughput, creation automated pipelines for model monitoring and retraining (MLOps), and testing integrations with intrusion prevention systems (IPS) to enable real-time and safe defensive actions.

7.6 Concluding Remarks

This dissertation provides empirical and systems evidence that deep sequence models, instantiated here as LSTM networks, constitute a viable and effective basis for network intrusion prediction. By combining a principled data pipeline, a high-performing multiclass predictor, and a deployable modular architecture, the work advances the state of practice toward proactive, data-driven cyber defence. Notwithstanding the limitations identified, the findings indicate a clear path toward more adaptive, explainable, and robust intrusion prediction systems capable of operating in evolving network environments.

Bibliography

[1] Martina Bet, “More than a third of UK schools hit by cripping cyber attacks with some hackers demanding hefty ransoms.” https://www.thesun.co.uk/news/34294097/schools-targeted-cyber-hackers-uk/, 2025. Accessed: 2025-04-08.

[2] Cloudflare, Inc., “About cloudflare.” https://www.cloudflare.com/, 2025. Accessed: October 14, 2025.

[3] Cloudflare, Inc., “Cloudflare documentation.” https://developers.cloudflare.com/, 2025. Accessed: October 14, 2025.

[4] Greg Otto, “Cloudflare detected (and blocked) the biggest DDoS attack on record.” https://cyberscoop.com/cloudflare-biggest-ddos-attack-mirai-variant-botnet/, 2025. Accessed: 2025-04-08.

[5] J. L. Leevy and T. M. Khoshgoftaar, “A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data,” *Journal of Big Data*, vol. 7, 2020.

[6] M. S. Ansari, V. Barto, and B. Lee, “Gru-based deep learning approach for network intrusion alert prediction,” *Future Generation Computer Systems*, vol. 128, pp. 235–247, 2022.

[7] University of New Brunswick, Canadian Institute for Cybersecurity, “Intrusion detection evaluation dataset (CIC-IDS2017).” https://www.unb.ca/cic/datasets/ids-2017.html. Accessed: 2025-09-18.

[8] R. Kaur, D. Gabrijeli, and T. Klobuar, “Artificial intelligence for cybersecurity: Literature review and future research directions,” *Information Fusion*, vol. 97, p. 101804, 2023.

[9] NIST, “Nist cybersecurity framework.” https://www.nist.gov/cyberframework, 2025. Accessed: October 14, 2025.

[10] Kaspersky, “What is cybercrime? How to protect yourself.” https://www.kaspersky.com/resource-center/threats/what-is-cybercrime, 2025. Accessed: 2025-07-12.

[11] IBM, "What is a cyberattack?." https://www.ibm.com/think/topics/cyber-attack, 2021. Accessed: 2025-07-12.

[12] Rapid7, "Vulnerabilities, Exploits, and Threats." https://www.rapid7.com/fundamentals/vulnerabilities-exploits-threats/, 2025. Accessed: 2025-07-12.

[13] Cisco, "What is a Hacker?." https://www.cisco.com/c/en/us/products/security/what-is-a-hacker.html, 2025. Accessed: 2025-07-12.

[14] IBM, "What is computer networking?." https://www.ibm.com/think/topics/networking, 2025. Accessed: 2025-07-12.

[15] Cisco, "What is a firewall?." https://www.cisco.com/site/us/en/learn/topics/security/what-is-a-firewall.html, 2025. Accessed: 2025-07-12.

[16] IBM, "What is an IDS?." https://www.ibm.com/think/topics/intrusion-detection-system, 2025. Accessed: 2025-07-12.

[17] IBM, "What is an IPS?." https://www.ibm.com/think/topics/intrusion-prevention-system, 2025. Accessed: 2025-07-12.

[18] Fortinet, "Top 20 most common types of cybersecurity attacks." https://www.fortinet.com/resources/cyberglossary/types-of-cyber-attacks, n.d. Accessed: 2024-12-05.

[19] Twingate, "What is a Man-in-the-Middle Attack? How It Works Examples." https://www.twingate.com/blog/glossary/man-in-the-middle%20attack. Accessed: 2025-09-18.

[20] Chris Power, "Session Hijacking Prevention: Everything You Should Know." https://powerconsulting.com/blog/session-hijacking-prevention/. Accessed: 2025-09-18.

[21] momrulhasan, "What is URL Interpretation Attack?." https://momrulhasan.medium.com/what-is-url-interpretation-attack-8a4f7df80db1. Accessed: 2025-09-18.

[22] imperva, "DNS Spoofing." https://www.imperva.com/learn/application-security/dns-spoofing/. Accessed: 2025-09-18.

[23] Justice Levine, "Brute-Force Attacks Uncovered: Types, Tools, and Defenses for Safety." https://guardiandigital.com/resources/blog/what-is-a-brute-force-attack. Accessed: 2025-09-18.

[24] Vitaly Unic, "8 Types of Web Application Attacks and Protecting Your Organization." https://brightsec.com/blog/8-types-of-web-application-attacks-and-protecting-your-organization/. Accessed: 2025-09-18.

[25] OWASP Foundation, “Sql injection.” https://owasp.org/www-community/attacks/SQL_Injection, 2025. Accessed: October 14, 2025.

[26] OWASP Foundation, “Cross site scripting (xss).” https://owasp.org/www-community/attacks/xss/, 2025. Accessed: October 14, 2025.

[27] imperva, “Parameter Tampering.” https://www.imperva.com/learn/application-security/parameter-tampering/. Accessed: 2025-09-18.

[28] H. Mohammadian, A. H. Lashkari, and A. Ghorbani, “Poisoning and evasion: Deep learning-based nids under adversarial attacks,” in *21st Annual International Conference on Privacy, Security and Trust (PST)*, 2024.

[29] geeksforgeeks, “What are Scanning Attacks?.” https://www.geeksforgeeks.org/ethical-hacking/what-are-scanning-attacks/. Accessed: 2025-09-18.

[30] mackeeper, “Malware vs Virus.” https://mackeeper.com/blog/malware-vs-virus/?srsltid=AfmBOoptCJ1fUF5yuf5iRmWlY6FQEf81AaBFIDrl6B6TjvXDvZa7RuGM. Accessed: 2025-09-18.

[31] A. D., J. S., P. Priya, Devipriya, P. Prithika, and V. R. E. Ramalingam, *Significance of Machine Learning and Deep Learning in Development of Artificial Intelligence*, pp. 25–44. 10 2022.

[32] A. Alhowaide, I. Alsmadi, and J. Tang, “Ensemble detection model for iot ids,” *Internet of Things*, vol. 16, p. 100435, 2021.

[33] A. Binbusayyis and T. Vaiyapuri, “Unsupervised deep learning approach for network intrusion detection combining convolutional autoencoder and one-class svm,” *Applied Intelligence*, vol. 51, no. 10, pp. 7094–7108, 2021.

[34] H. Albasheer, M. Md Siraj, A. Mubarakali, O. Elsier Tayfour, S. Salih, M. Hamdan, S. Khan, A. Zainal, and S. Kamarudeen, “Cyber-attack prediction based on network intrusion detection systems for alert correlation techniques: A survey,” *Sensors*, vol. 22, no. 4, 2022.

[35] A. Malekian and N. Chitsaz, “Chapter 4 - concepts, procedures, and applications of artificial neural network models in streamflow forecasting,” in *Advances in Streamflow Forecasting* (P. Sharma and D. Machiwal, eds.), pp. 115–147, Elsevier, 2021.

[36] Annie Badman and Matthew Kosinski, “What is a dataset?.” https://www.ibm.com/think/topics/dataset, 2024. Accessed: 2025-04-18.

[37] Bharath K, “Introduction to deep neural networks.” https://www.datacamp.com/tutorial/introduction-to-deep-neural-networks, 2023. Accessed: 2024-12-14.

[38] Datacamp, “An introduction to convolutional neural networks (cnns).” https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns, 2023. Accessed: 2024-12-11.

[39] Thejas Kiran, “Unlocking the potential of convolutional neural networks (cnns) in time series forecasting.” https://thejaskiran99.medium.com/unlocking-the-potential-of-convolutional-neural-networks-cnns-in-time-series-forecasting-b2fac329e184, 2023. Accessed: 2024-12-11.

[40] Geeks for Geeks, “Introduction to recurrent neural networks.” https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/, 2024. Accessed: 2024-12-14.

[41] Gaudenz Boesch, “Deep belief networks (dbns) explained.” https://viso.ai/deep-learning/deep-belief-networks/, 2024. Accessed: 2024-12-14.

[42] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[43] F. Gers, N. Schraudolph, and J. Schmidhuber, “Learning precise timing with lstm recurrent networks,” *Journal of Machine Learning Research*, vol. 3, pp. 115–143, 01 2002.

[44] S. Kannan, K. Subbaram, and M. Faiyazuddin, “Chapter 17 - artificial intelligence in vaccine development: Significance and challenges ahead,” in *A Handbook of Artificial Intelligence in Drug Delivery* (A. Philip, A. Shahiwala, M. Rashid, and M. Faiyazuddin, eds.), pp. 467–486, Academic Press, 2023.

[45] C. Dave Bergmann, “What is an autoencoder?.” https://www.ibm.com/think/topics/autoencoder, 2023. Accessed: 2024-12-16.

[46] Python, “What is Python? Executive Summary.” https://www.python.org/doc/essays/blurb/, 2025. Accessed: 2025-07-12.

[47] NumPy Developers, “Numpy: The fundamental package for scientific computing with python.” https://numpy.org/, 2025. Accessed: October 14, 2025.

[48] SciPy Developers, “Scipy: Fundamental algorithms for scientific computing in python.” https://scipy.org/, 2025. Accessed: October 14, 2025.

[49] Pandas Development Team, “Pandas: Python data analysis library.” https://pandas.pydata.org/, 2025. Accessed: October 14, 2025.

[50] IBM, “What is Scikit-Learn (Sklearn)?.” https://www.ibm.com/think/topics/scikit-learn, 2025. Accessed: 2025-07-12.

[51] Nvidia, “TensorFlow.” https://www.nvidia.com/en-eu/glossary/tensorflow/, 2025. Accessed: 2025-07-12.

[52] University of New Brunswick | UNB, "CICFlowMeter (formerly ISCXFlowMeter)." https://www.unb.ca/cic/research/applications.html, 2025. Accessed: 2025-07-12.

[53] IBM, "What is Redis?." https://www.ibm.com/think/topics/redis, 2025. Accessed: 2025-07-12.

[54] AWS, "InfluxDB on AWS - Fully Managed InfluxDB Databases." https://aws.amazon.com/influxdb/, 2025. Accessed: 2025-07-12.

[55] AWS, "What is Docker?." https://aws.amazon.com/docker/, 2025. Accessed: 2025-07-12.

[56] Docker, "Docker Compose." https://docs.docker.com/compose/, 2025. Accessed: 2025-07-12.

[57] Canonical, "Ubuntu." https://ubuntu.com/, 2025. Accessed: 2025-07-12.

[58] Cybrary, "What is Kali Linux?." https://www.cybrary.it/blog/a-brief-introduction-to-kali-linux-for-cyber-security, 2025. Accessed: 2025-07-12.

[59] University of California, Irvine, "KDD Cup 1999 Data." https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999. Accessed: 2025-09-18.

[60] Lincoln Laboratory, MIT, "2000 DARPA Intrusion Detection Scenario Specific Datasets." https://www.ll.mit.edu/r-d/datasets/2000-darpa-intrusion-detection-scenario-specific-datasets. Accessed: 2025-09-18.

[61] University of New Wales Sydney, "The UNSW-NB15 Dataset." https://research.unsw.edu.au/projects/unsw-nb15-dataset. Accessed: 2025-09-18.

[62] R. Kimanzi, P. Kimanga, D. Cherori, and P. K. Gikunda, "Deep learning algorithms used in intrusion detection systems - a review," *ArXiv*, vol. abs/2402.17020, 2024.

[63] Mendeley Data, "Dataset of intrusion detection alerts from a sharing platform." https://data.mendeley.com/datasets/p6tym3fghz/1, 2019. Accessed: 2025-09-18.

[64] R. Nassr, f. saeed, and A. A.A., "Enhancing intrusion detection systems using a deep learning and data augmentation approach," *Systems*, vol. 12, 03 2024.

[65] P. Kumar, J. Liu, A. S. Md Tayeen, S. Misra, H. Cao, J. Harikumar, and O. Perez, "Flnet2023: Realistic network intrusion detection dataset for federated learning," in *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*, pp. 345–350, 2023.

[66] S. Samarakoon, Y. Siriwardhana, P. Porambage, M. Liyanage, S.-Y. Chang, J. Kim, J. Kim, and M. Ylianttila, "5g-nidd: A comprehensive network intrusion detection dataset generated over 5g wireless network," 2022.

[67] J. Jose and D. Jose, "Deep learning algorithms for intrusion detection systems in internet of things using cic-ids 2017 dataset," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 13, p. 1134, 02 2023.

[68] A. Rosay, E. Cheval, F. Carlier, and L. Pascal, "Network intrusion detection: A comprehensive analysis of cic-ids2017," 02 2022.

[69] A. ROSAY, F. CARLIER, E. CHEVAL, and P. LEROUX, "From cic-ids2017 to lycos-ids2017: A corrected dataset for better performance," in *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, WI-IAT '21, (New York, NY, USA), p. 570575, Association for Computing Machinery, 2022.

[70] B. Imene, D. Aissa, and D. Rafik, "Analyzing and exploring cic-ids 2017 dataset," *International Journal of Research Studies in Computer Science and Engineering*, vol. 9, pp. 10–15, 01 2023.

[71] S. Panwar, Y. Raiwani, and L. Panwar, "An intrusion detection model for cicids-2017 dataset using machine learning algorithms," pp. 1–10, 11 2022.

[72] Z. I. Khan, M. Afzal, and K. Shamsi, "A comprehensive study on cic-ids2017 dataset for intrusion detection systems," 02 2024.

[73] B. Reis, E. Maia, and I. Praça, *Selection and Performance Analysis of CICIDS2017 Features Importance*, pp. 56–71. 04 2020.

[74] K. Dhanya, S. Vajipayajula, K. Srinivasan, A. Tibrewal, T. S. Kumar, and T. G. Kumar, "Detection of network attacks using machine learning and deep learning models," *Procedia Computer Science*, vol. 218, pp. 57–66, 2023. International Conference on Machine Learning and Data Engineering.

[75] S. J. Moore, F. Cruciani, C. D. Nugent, S. Zhang, I. Cleland, and S. Sani, "Deep learning for network intrusion: A hierarchical approach to reduce false alarms," *Intelligent Systems with Applications*, vol. 18, p. 200215, 2023.

[76] O. A. Alzubi, I. Qiqieh, and J. A. Alzubi, "Fusion of deep learning based cyberattack detection and classification model for intelligent systems," *Cluster Computing*, vol. 26, no. 2, pp. 1363–1374, 2023.

[77] J. Zhang, L. Pan, Q.-L. Han, C. Chen, S. Wen, and Y. Xiang, "Deep learning based attack detection for cyber-physical system cybersecurity: A survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 3, pp. 377–391, 2022.

[78] A. H. I. E. Tariq, M. B. I. E. Tariq, and S. Lu, "Hybrid ai-driven techniques for enhancing zeroday exploit detection in intrusion detection system (ids)," pp. 156–160, 09 2024.

[79] V. Kurnala, S. Naik, D. Surapaneni, and C. Reddy, “Hybrid detection: Enhancing network server intrusion detection using deep learning,” pp. 248–251, 10 2023.

[80] T. Sowmya and E. Mary Anita, “A comprehensive review of ai based intrusion detection system,” *Measurement: Sensors*, vol. 28, p. 100827, 2023.

[81] Y. Wu, B. Zou, and Y. Cao, “Current status and challenges and future trends of deep learning-based intrusion detection models,” *Journal of Imaging*, vol. 10, no. 10, 2024.

[82] T. Yi, X. Chen, Y. Zhu, W. Ge, and Z. Han, “Review on the application of deep learning in network attack detection,” *Journal of Network and Computer Applications*, vol. 212, p. 103580, 2023.

[83] D. S. Kim and J. S. Park, “Network-based intrusion detection with support vector machines,” in *Information Networking* (H.-K. Kahng, ed.), (Berlin, Heidelberg), pp. 747–756, Springer Berlin Heidelberg, 2003.

[84] H. Wang, J. Gu, and S. Wang, “An effective intrusion detection framework based on svm with feature augmentation,” *Knowledge-Based Systems*, vol. 136, pp. 130–139, 2017.

[85] W.-C. Lin, S.-W. Ke, and C.-F. Tsai, “Cann: An intrusion detection system based on combining cluster centers and nearest neighbors,” *Knowledge-Based Systems*, vol. 78, pp. 13–21, 2015.

[86] B. S. Sharmila and R. Nagapadma, “Intrusion detection system using naive bayes algorithm,” in *2019 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*, pp. 1–4, 2019.

[87] M. Belouch, S. El Hadaj, and M. Idhammad, “Performance evaluation of intrusion detection based on machine learning using apache spark,” *Procedia Computer Science*, vol. 127, pp. 1–6, 2018. PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON INTELLIGENT COMPUTING IN DATA SCIENCES, ICDS2017.

[88] Akashdeep, I. Manzoor, and N. Kumar, “A feature reduced intrusion detection system using ann classifier,” *Expert Systems with Applications*, vol. 88, pp. 249–257, 2017.

[89] P. Liu, “An intrusion detection system based on convolutional neural network,” in *Proceedings of the 2019 11th International Conference on Computer and Automation Engineering*, ICCAE 2019, (New York, NY, USA), p. 6267, Association for Computing Machinery, 2019.

[90] S. Nayyar, S. Arora, and M. Singh, “Recurrent neural network based intrusion detection system,” in *2020 International Conference on Communication and Signal Processing (ICCSP)*, pp. 0136–0140, 2020.

[91] S. Sivamohan, S. Sridhar, and S. Krishnaveni, "An effective recurrent neural network (rnn) based intrusion detection via bi-directional long short-term memory," in *2021 International Conference on Intelligent Technologies (CONIT)*, pp. 1–5, 2021.

[92] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, p. 21954 21961, 2017. Cited by: 1400; All Open Access, Gold Open Access.

[93] F. Farahnakian and J. Heikkonen, "A deep auto-encoder based approach for intrusion detection system," in *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pp. 178–183, 2018.

[94] H. Zhang, L. Huang, C. Q. Wu, and Z. Li, "An effective convolutional neural network based on smote and gaussian mixture model for intrusion detection in imbalanced dataset," *Computer Networks*, vol. 177, p. 107315, 2020.

[95] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, "Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2018.

[96] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1222–1228, 2017.

[97] M. A. Ferrag, L. Maglaras, H. Janicke, and R. Smith, "Deep learning techniques for cyber security intrusion detection : A detailed analysis," 09 2019.

[98] S. Srinivasan, S. Ketha, V. Ravi, A. Soman, and S. Kp, "Towards evaluating the robustness of deep intrusion detection models in adversarial environment," 01 2020.

[99] K. Jakotiya, V. Shirsath, and R. Mishra, "Intrusion detection system using deep learning and machine learning: Review," pp. 1–4, 11 2023.

[100] X. Song, M. Song, and X. Li, "A computer network security intrusion detection algorithm based on deep learning," pp. 472–476, 07 2024.

[101] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Applied Sciences*, vol. 9, p. 4396, 10 2019.

[102] Canadian Institute for Cybersecurity, "Canadian institute for cybersecurity." `CanadianInstituteforCybersecurity`, 2025. Accessed: October 14, 2025.

[103] A. B. Hassanat, A. S. Tarawneh, G. A. Altarawneh, and A. Almuhaimeed, "Stop oversampling for class imbalance learning: A critical review," 2022.

[104] P. A. A. Resende and A. C. Drummond, "A survey of random forest based methods for intrusion detection systems," *ACM Comput. Surv.*, vol. 51, May 2018.

[105] IBM, "What is multicollinearity? ." https://www.ibm.com/think/topics/multicollinearity. Accessed: 2025-09-18.

[106] Z. Luna, "Feature selection in machine learning: Correlation matrix | univariate testing | rfecv." https://medium.com/geekculture/feature-selection-in-machine-learning-correlation-matrix-univariate-testing-rfe 2021. Accessed: 2025-5-10.

[107] M. Rosenblatt, "Remarks on a multivariate transformation," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 470–472, 1952.

[108] A. W. v. d. Vaart, *Asymptotic Statistics*. Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press, 1998.

[109] geeksforgeeks, "Quantiletransformer using scikit-learn." https://www.geeksforgeeks.org/quantiletransformer-using-scikit-learn/, 2024. Accessed: 2025-5-10.

[110] imbalanced learn, "SMOTE." https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html, 2025. Accessed: 2025-05-12.

[111] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Intell. Res. (JAIR)*, vol. 16, pp. 321–357, 06 2002.

[112] DeepAI, "One-Hot Encoding." https://deepai.org/machine-learning-glossary-and-terms/one-hot-encoding, 2025. Accessed: 2025-05-12.