UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

UNSUPERVISED MACHINE LEARNING FOR REAL-TIME NETWORK INTRUSION DETECTION: NETWORK IMPLEMENTATION, ATTACK SIMULATION AND AI ENCHANCEMENT

Diploma Thesis

Moysis Moysis

Supervisor**:** Antonopoulos Christos

July 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ ΧΩΡΙΣ ΕΠΙΒΛΕΨΗ ΓΙΑ ΑΝΙΧΝΕΥΣΗ ΕΙΣΒΟΛΩΝ ΣΕ ΔΙΚΤΥΑ ΣΕ ΠΡΑΓΜΑΤΙΚΟ ΧΡΟΝΟ: ΥΛΟΠΟΙΗΣΗ ΔΙΚΤΥΟΥ, ΠΡΟΣΟΜΟΙΩΣΗ ΕΠΙΘΕΣΕΩΝ ΚΑΙ ΒΕΛΤΙΩΣΗ ΑΛΓΟΡΙΘΜΩΝ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ

Διπλωματική Εργασία

Μωυσής Μωυσής

Επιβλέπων: Αντωνόπουλος Χρήστος

Ιούλιος 2025

Approved by the Examination Committee:

Supervisor **Antonopoulos Christos**

Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Lalis Spyros**

Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Mpellas Nikolaos**

Professor, Department of Electrical and Computer Engineering, University of Thessaly

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

The declarant

Moysis Moysis

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελούν αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλουν οποιασδήποτε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχουν έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Δηλώνω επίσης ότι τα αποτελέσματα της εργασίας δεν έχουν χρησιμοποιηθεί για την απόκτηση άλλου πτυχίου. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο/Η Δηλών/ούσα

Μωυσής Μωυσής

Diploma Thesis

Unsupervised Machine Learning for Real-time Network Intrusion Detection: Network Implementation, Attack Simulation and AI Enhancement

Moysis Moysis

ABSTRACT

This thesis develops a machine learning system which detects network traffic intrusions in real-time through autoencoder applications. The traditional intrusion detection systems (IDS) depend on signature-based methods which fail to protect against new and unknown attacks. The thesis applies unsupervised learning techniques to discover unusual network patterns without needing attack signature information.

The proposed system uses live packet data capture to build flow-based features which then feed into an autoencoder-based anomaly detection system that uses reconstruction errors for identification. The system uses a lightweight architecture that allows real-time processing and immediate response capabilities including automatic IP blocking through system-level firewall rules. The thesis implements dual autoencoder architecture (AnomalyDAE) alongside supervised learning with LightGBM to evaluate detection performance across different strategies.

The evaluation uses benchmark datasets (UNSW-NB15 and Kyoto 2006+) to analyze accuracy and precision and recall and false positive rates in detail. The unsupervised autoencoder served as the main model because it learns normal network patterns without attack labels which makes it suitable for detecting unknown zero-day attacks. The system demonstrates how AI-based cybersecurity solutions can be deployed for real-time monitoring and proactive defense

Keywords: Intrusion Detection System, Zero-day Attack, Autoencoder, Reconstruction Error, Anomaly Detection, Real-Time Security, Cybersecurity, Network Traffic

Διπλωματική Εργασία

Μηχανική Μάθηση Χωρίς Επίβλεψη Για Ανίχνευση Εισβολών σε Δίκτυα σε Πραγματικό Χρόνο: Υλοποίηση Δικτύου, Προσομοίωση Επιθέσεων και Βελτίωση Αλγόριθμων Τεχνητής Νοημοσύνης

Μωυσής Μωυσής

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία παρουσιάζει τον σχεδιασμό, την ανάπτυξη και την αξιολόγηση ενός συστήματος ανίχνευσης εισβολών σε πραγματικό χρόνο με χρήση τεχνικών μηχανικής μάθησης και συγκεκριμένα autoencoders. Τα παραδοσιακά συστήματα ανίχνευσης εισβολών (IDS) βασίζονται κυρίως σε μεθόδους με υπογραφές, οι οποίες αδυνατούν να ανιχνεύσουν άγνωστες ή καινούργιες επιθέσεις. Αντιθέτως, η παρούσα εργασία αξιοποιεί τη μη επιβλεπόμενη μάθηση (unsupervised learning) για τον εντοπισμό ανωμαλιών στη συμπεριφορά του δικτύου, χωρίς την ανάγκη προηγούμενης γνώσης.

Το προτεινόμενο σύστημα καταγράφει πακέτα δικτύου σε πραγματικό χρόνο, εξάγει χαρακτηριστικά ροών και εφαρμόζει έναν autoencoder για την ανίχνευση ανωμαλιών βάσει του σφάλματος ανακατασκευής. Η αρχιτεκτονική του συστήματος είναι ελαφριά και επεκτάσιμη, επιτρέποντας άμεση επεξεργασία και αντίδραση, όπως τον αποκλεισμό ύποπτων διευθύνσεων IP μέσω iptables. Επιπλέον, υλοποιούνται και συγκρίνονται επιπλέον μέθοδοι όπως το LightGBM (supervised learning) και ο Dual Autoencoder (AnomalyDAE) για την αξιολόγηση διαφορετικών στρατηγικών ανίχνευσης.

Η πειραματική αξιολόγηση βασίζεται σε δύο πρότυπα σύνολα δεδομένων (UNSW-NB15 και Kyoto 2006+), με λεπτομερή ανάλυση δεικτών απόδοσης όπως ακρίβεια, ευαισθησία, ανάκληση και ποσοστά ψευδών συναγερμών. Επιλέχθηκε η χρήση ενός μη επιβλεπόμενου autoencoder ως βασικό μοντέλο λόγω της ικανότητάς του να μαθαίνει τα φυσιολογικά μοτίβα συμπεριφοράς του δικτύου χωρίς την ανάγκη επισημασμένων δεδομένων επιθέσεων, καθιστώντας το ιδιαίτερα κατάλληλο για την ανίχνευση άγνωστων ή zero-day επιθέσεων. Τα αποτελέσματα δείχνουν ότι ο Dual Autoencoder προσφέρει καλύτερη γενίκευση για την ανίχνευση ανωμαλιών, ενώ το LightGBM επιτυγχάνει υψηλότερη ακρίβεια στην ταξινόμηση επιθέσεων με επισημασμένα δεδομένα. Το σύστημα αποδεικνύει την πρακτική δυνατότητα υλοποίησης λύσεων κυβερνοασφάλειας βασισμένων στην τεχνητή νοημοσύνη, με δυνατότητα ταυτόχρονης παρακολούθησης και προληπτικής άμυνας σε πραγματικό χρόνο.

Λέξεις κλειδιά: Σύστημα Ανίχνευσης Εισβολών, Καινούργια Επίθεση, Autoencoder, Σφάλμα Ανακατασκευής, Ανίχνευση Ανωμαλιών, Ασφάλεια σε Πραγματικό Χρόνο, Κυβερνοασφάλεια, Δικτυακή Κίνηση

ACKNOWLEDGEMENTS

Reaching the ending of this thesis marks not just the conclusion of a research project, but the culmination of an unforgettable five-year journey as a student of Electrical and Computer Engineering. These years were filled with challenges, growth, learning and moments that shaped me both personally and professionally.

I would like to express my sincere gratitude to my supervisor Christos Antonopoulos for his trust, guidance and support throughout the thesis process. His advice and encouragement were invaluable in helping me carry this project to completion.

I would also like to thank my family for standing by me over the course of these years. Their ongoing support made it possible for me to focus on my studies without additional pressure. I'm especially grateful to my father and mother for their many sacrifices and for always providing a stable and supportive environment that allowed me to grow.

I want to express my appreciation to all my friends who made these years more enjoyable. Whether through shared struggles and assignments, group projects, late-night study sessions, or much-needed breaks, your company was an essential part of this experience.

Also, I would like to say a big thank you to Nikitopoulos Giorgos for his valuable help to this thesis.

Lastly, Rafaela deserves special thanks for offering a steady presence throughout these years. Her presence brought clarity when things felt overwhelming and perspective when it was most needed. Through quiet strength and thoughtful support, she made even the hardest days more manageable, and the good ones more meaningful.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Digital communication functions as a fundamental element in both personal and professional environments of our interconnected world. The growth of cloud storage and online banking and enterprise-level IT systems expands the attack surface which becomes more vulnerable to cyber threats. The growth rate of this expansion is truly remarkable. The world generated 328.77 million terabytes of data daily during 2023 [1]. The financial impact of cybercrime reached more than 8 trillion dollars in 2023 which demonstrates the substantial economic consequences of digital security threats [2]. The cyberattacks on T-Mobile and MGM Resorts exposed millions of user sensitive data in real-world incidents. The recent data breaches demonstrate why organizations need cybersecurity systems that combine strength with intelligent adaptation capabilities to handle emerging threats [3] [4].

Traditional Intrusion Detection Systems (IDS) that depend on signature-based detection methods have become insufficient for protecting against modern cyber threats that evolve quickly. The systems operate by comparing network traffic against established databases of recognized attack signatures. The systems prove successful against known exploits yet fail to detect zero-day attacks and new intrusion techniques that avoid established patterns [5], [6]. Sophisticated attackers use payload obfuscation techniques and polymorphic methods which make static rule-based detection systems unreliable.

The challenges in cybersecurity have been addressed through the implementation of Artificial Intelligence (AI) and Machine Learning (ML) as transformative cybersecurity tools. AI-powered systems learn from data and adapt over time to detect subtle anomalies which traditional methods cannot identify [5]. AI-driven IDS systems analyze large volumes of network traffic in real-time to create behavioral models that detect abnormal patterns which could signal a security breach. The proactive method enhances the speed at which threats can be addressed.

The 2023 IBM report shows that organizations implementing AI-based cybersecurity tools decreased their breach detection time by 96 days and reduced their average incident costs to $3.5 million [7]. AI integration enables automated threat hunting and response which decreases human analyst workload and enhances system resilience. The evolution of cyberattacks toward automation and targeted attacks requires defensive tools to develop similar intelligence capabilities for effective protection in advanced threat environments.

1.1 THE URGENCY OF INTELLIGENT INTRUSION DETECTION

The cybersecurity environment undergoes significant transformations at present. Cyberattacks have become more frequent and larger in scale and complexity because threat actors have become more sophisticated, while Actors have become more sophisticated,

while Malware-as-a-Service (MaaS), ransomware toolkits and automated botnets have spread throughout the network [8]. These technologies allow unskilled attackers to execute destructive attacks which overwhelm established security systems [9]. Organizations must move beyond reactive and static defense approaches because the evolving threat environment demands new security measures [7].

Organizations need immediate implementation of real-time intrusion detection systems which detect emerging threats in real-time while taking autonomous actions to stop damage progression.

The research has become especially relevant because technology has reached a point where it meets operational requirements. Real-time deployment of machine learning models in high-throughput network environments was previously limited by hardware constraints and insufficient data and complex models. Today, these constraints are rapidly diminishing. The combination of high-performance computing hardware with optimized deep learning frameworks and parallelized data processing techniques enables the deployment of models that analyze live network traffic streams in real time.

The development of unsupervised learning techniques, especially autoencoders, has enabled powerful methods for anomaly detection [10]. These models demonstrate exceptional ability to discover normal network patterns which enable them to detect minor deviations without requiring large amounts of labeled attack data. This approach proves highly beneficial for real-world applications because zero-day attacks and new threats are becoming more prevalent while labeled datasets remain scarce or incomplete.

The research combines three essential elements. The consistently increasing cyber threats, technological progress and the practical challenges organizations face. The research uses modern AI technologies to create operational systems which support strategic goals while addressing current organizational constraints.

1.2 THESIS OBJECTIVE

The main purpose of this thesis involves creating and testing an intelligent intrusion detection system (IDS) that detects network traffic anomalies through real-time machine learning operations. The system unites unsupervised anomaly detection methods with supervised classification techniques to protect against known and unknown digital threats in dynamic environments. Our research focuses on achieving the following specific objectives:

- **Improved Intrusion Detection Accuracy in Evolving Threat Landscapes**: Traditional IDS struggle to identify zero-day attacks and new patterns of cyber threats. This research presents AI-powered anomaly detection which uses learned

normal network behavior to identify unknown threats thereby enhancing detection capabilities in operational environments.

- **Real-Time Response to Cyber Threats**: The system enables iptables firewall integration with live anomaly detection to automatically block malicious IP addresses immediately which reduces reaction times and helps prevent major breaches from becoming more severe.
- **Reduction of Human Burden through Intelligent Automation**: Cybersecurity teams experience alert fatigue because they receive numerous false-positive alerts. The unsupervised approach in this thesis learns detailed patterns from real data which reduces false alarms so human analysts can concentrate on critical threats.
- **Adaptable to Diverse Network Environment**: The models demonstrated effectiveness through testing with synthetic (UNSW-NB15) and real-world (Kyoto 2006+) datasets which confirmed their adaptability. This method applies to all network environments including enterprise networks as well as cloud infrastructures and small-scale IoT networks.
- **Reduction in Breach Costs and Downtime**: IBM reports that AI-based detection systems decreased breach detection time to 96 days while reducing average incident costs to $3.5 million [7]. The system implements anomaly detection with automated mitigation that helps organizations decrease their losses according to the research findings.
- **Proactive Defense Against AI-Powered Attacks**: The work shows how defensive AI capabilities can match up with machine learning attacks through its demonstration of defensive AI evolution. The research establishes a framework to bridge theoretical security tool development with operational security implementation.

The thesis achieves its objectives by developing a deployable anomaly-based network intrusion detection system that demonstrates machine learning applications for improving cybersecurity defense systems.

1.3 THESIS CONTRIBUTION

This thesis tackles immediate cybersecurity challenges affecting modern digital infrastructure while delivering applicable solutions for real-world implementation.

- We aim to create an autoencoder detection pipeline which learns from normal network behavior patterns to detect anomalies through reconstruction errors. In addition, we develop a Gradient Boosting supervised classifier which learns from labeled attack data to classify the detected anomalies from the autoencoder.
- We evaluate the autoencoder detection performance for network intrusions by using UNSW-NB15 and Kyoto 2006+ datasets as real-world data sources

- The Thesis includes complete data preprocessing steps which involve normalization, categorical encoding and class balancing for the attack types with less data.
- The system implements Linux iptables to establish real-time response capabilities that automatically block IP addresses showing suspicious network activity. This way, it achieves practical application through real-time traffic capture, flow-based feature extraction and detection model deployment with low latency.

Our work presents practical contributions that demonstrate its significance for modern cybersecurity challenges in the rapidly changing digital security environment. The Thesis connects advanced machine learning methods with real-time system-level integration to both improve academic knowledge of anomaly-based intrusion detection and deliver a functional solution that solves operational requirements. The proposed system demonstrates how intelligent automation can enhance digital defense strategies while reducing organizational risk and enabling scalable proactive responses to current and emerging cyber threats.

1.4 THESIS STRUCTURE

The entire thesis contains eight chapters which establish a progressive sequence from essential concepts to the completed intrusion detection system. In **Chapter 2** we provide fundamental theoretical and technical information, which establishes the foundation for this thesis. This chapter discusses the basics of machine learning as well as learning paradigms, optimization approaches, evaluation methods and autoencoder applications in anomaly detection. **Chapter 3** explores the relevant research literature through a review of machine learning-based intrusion detection systems along with an identification of system limitations and an explanation of this study's position in the research domain. In **Chapter 4** we define the datasets together with a description of the feature engineering procedures. The research uses the UNSW-NB15 and Kyoto 2006+ datasets and explains preprocessing techniques while explaining how features were selected and engineered for model training. In **Chapter 5** methodology and the complete system architecture of the proposed approach is explained. The paper describes the combination of unsupervised and supervised models, while detailing data collection processes and packet processing methods along with thresholding approaches and system integration through iptables. **Chapter 6** provides operational details regarding model training, inference setup and live traffic monitoring together with response mechanisms. Technical problems and engineering decisions that emerged during development are demonstrated throughout this chapter. **Chapter 7** demonstrates the proposed method results. The study compares performance metrics among models while evaluating real-time detection success rates and presenting findings against established IDS tools. Lastly, **Chapter 8** synthesizes essential thesis results before

discussing limitations and proposing future research directions that align with production environment requirements.

The thesis contains additional appendices with detailed feature definitions together with system configuration information and code examples. The bibliography contains all sources which contributed to this research.

CHAPTER 2: BACKGROUND

The creation of contemporary intrusion detection systems (IDS) requires knowledge across multiple related fields which span from machine learning mathematical principles to real-time traffic monitoring and system integration practices. The evolution of cyberattacks necessitates corresponding advancements in defensive mechanisms which unite theoretical precision with engineering productivity. The following chapter establishes the fundamental understanding of concepts, models and technologies which serve as the foundation for this thesis.

Machine learning through unsupervised learning methods helps detect anomalies in tasks where attack data lacks labels or is scarce. The traditional rule-based IDS methods fail to detect zero-day attacks because these attacks do not match any existing patterns. Autoencoders and other unsupervised models detect potential threats by learning normal network behavior patterns to identify deviations. The successful deployment of these models requires complete knowledge about their operational mechanisms including architectural designs, training approaches and performance boundaries. The correct implementation and evaluation of these models depends on understanding feature learning and dimensionality reduction techniques as well as reconstruction loss and threshold tuning concepts. Equally important is the understanding of the cybersecurity context in which these models operate. Concepts such as network traffic flows, packet structure, protocol behavior (e.g., TCP, UDP), and common attack vectors provide the operational domain where anomaly detection is applied. Without this context, it is impossible to extract meaningful features from raw traffic or to interpret the results of a machine learning model accurately. To provide this necessary foundation, the current chapter introduces the machine learning concepts relevant to this work, including supervised and unsupervised learning, anomaly detection principles, and autoencoder architecture. It also reviews the metrics used to evaluate detection models, such as accuracy, precision, recall, and F1-score.

2.1 MACHINE LEARNING IN GENERAL

Machine Learning (ML) represents a subfield of artificial intelligence which develops algorithms and statistical models to let computers learn from data instead of receiving explicit programming. Machine learning systems differ from traditional software development because they learn patterns and behaviors from historical observations instead of using manually defined rules by developers. The ability of ML to handle complex domains like network security becomes highly effective because it adapts to changing threats and unpredictable patterns.

Machine learning achieves its core purpose by developing models that effectively apply past data to predict or classify new inputs which have not been seen before. The

development of a machine learning system requires a multi-stage pipeline which contains essential stages that determine the final accuracy of the model:

- Data collection: The first stage requires obtaining substantial amounts of unprocessed data which directly relates to the problem domain. The process may require combining various datasets to enhance the available information while improving model generalization capabilities. Network intrusion detection data collection includes network traffic logs together with packet capture (PCAP) files and telemetry data from firewalls and intrusion detection systems (IDS) and other network monitoring tools.
- Preprocessing: The stage where raw data is cleaned and transformed into a structured and consistent format. This step may involve handling missing values, converting timestamps, normalizing numerical values, encoding categorical variables (such as protocols or states) and filtering out irrelevant or noisy data. The goal is to ensure that the input data is ready for model training without introducing bias or inconsistency.
- Feature selection: Aims to identify or construct the most informative variables from the dataset. Rather than using all available fields, which can include redundant or irrelevant information, this step reduces the dimensionality and improves model efficiency. In cybersecurity, useful features may include the number of packets per connection, total bytes transferred, connection duration, TTL values or flow rates.
- Model training: It is the process of feeding the preprocessed feature data into a machine learning model, so it can learn the statistical relationships and patterns in the training data by minimizing an objective function, such as classification loss or reconstruction error. During this stage, internal parameters are updated iteratively using optimization algorithms (e.g gradient descent).
- Validation and testing: Ensures that the trained model performs well not only on the training data but also on previously unseen data. Validation helps in tuning hyperparameters and avoiding overfitting, while testing provides an unbiased estimate of the model's generalization ability. Common evaluation metrics include accuracy, precision, recall, F1-score and confusion matrices – all of which are discussed later in this chapter.
- Deployment: The last stage of machine learning pipeline implementation involves deploying a trained and validated model into operational systems to perform real-time inference on live data. The model integration process in cybersecurity would require embedding it into network monitoring tools which analyze real-time traffic to perform actions like logging anomalies, generating alerts and blocking suspicious IP addresses. The thesis does not include complete production deployment, but its system design and implementation followed deployment feasibility principles. The system development focuses on real-time processing

capabilities and modular system architecture and lightweight model inference because these features are necessary for future operational security infrastructure integration.

A critical aspect of any machine learning system is its ability to generalize. Generalization is the ability to make accurate predictions on previously unseen cases. This involves balancing underfitting with overfitting. Underfitting is the situation where the model is too simplistic to capture meaningful patterns in the data, while in overfitting the model captures noise or anomalies from the dataset and fails to perform well with new inputs. After model evaluation, the pipeline often returns to earlier stages (e.g., feature engineering or retraining) in an iterative process to improve performance.

2.2 TYPES OF MACHINE LEARNING

The main categories of machine learning techniques consist of supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning. The learning paradigms differ from each other through their data interpretation methods and their training feedback reception. The selection of learning type depends on the amount of labeled data available and the characteristics of the task and the desired system behavior. The design of intelligent systems requires knowledge about each learning paradigm's features together with its advantages and constraints. The following subsections deliver an extensive explanation of each machine learning type.

2.2.1 Supervised machine learning

Supervised learning is one of the most established and widely used paradigms of machine learning. In this approach, models are trained on labeled datasets, where each input sample is associated with a known output. The goal for the model is to learn a mapping algorithm from inputs to outputs that generalize well with unseen data. There are two main categories of supervised learning:

- **Regression**: Focuses on predicting continues target variables that represent numerical values. These algorithms learn the pattern between input features and a continues outcome, so they can estimate values within a range. For example, regression can be used to estimate the price of a house depending on its size and location.
- **Classification**: It involves predicting categorical target variables that represent distinct classes or labels. It is used to assign input data to one of several predefined

categories based on learning patterns from the data. For example, classification can be applied to determine if an email is spam or not spam.

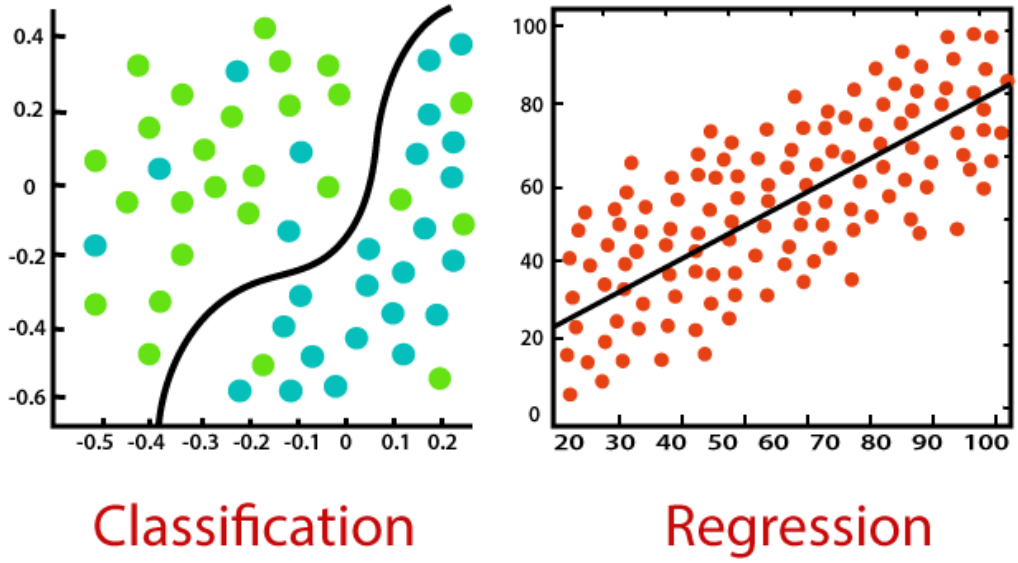

Figure 2.1: Visual comparison between classification and regression. [32]

The supervised machine learning field contains multiple algorithms which operate under distinct assumptions while employing different learning methods for various applications. The following list presents an overview of the most used algorithms:

- **Logistic Regression**: Linear classification algorithm logistic regression predicts the probability of an input belonging to a specific class. The algorithm uses a sigmoid function to establish binary output label relationships with input features while producing output values between 0 and 1. Logistic regression achieves good results when data separates linearly and when interpretable decision boundaries are needed.
- **Neural Networks**: Composed of layers of interconnected nodes, neural networks can model complex nonlinear relationships. They are highly effective in tasks involving large datasets and unstructured data like images, audio, or text.
- **Decision Trees**: Decision trees are non-parametric models that split the data based on feature thresholds to form a tree-like structure of decisions. Each node of the tree represents a condition, and each leaf node represents a predicted class as output.
- **Random Forest**: Random forest is an ensemble learning method that requires multiple decision trees on different subsets of the data and averages their prediction. It improves accuracy and avoids overfitting compared to individual decision trees.

- **Support Vector Machines (SVMs)**: The Support Vector Machine classifier searches for the best hyperplane that separates different classes in the feature space. The kernel functions enable the classifier to handle non-linearly separable data.
- **Gradient Boosting**: It is an ensemble learning method that builds a strong predictive model by sequentially combining multiple weak learners, typically decision trees. Each new model is trained to correct the errors of the previous one by minimizing a chosen loss function. This approach allows the final model to capture complex patterns and achieve high accuracy.

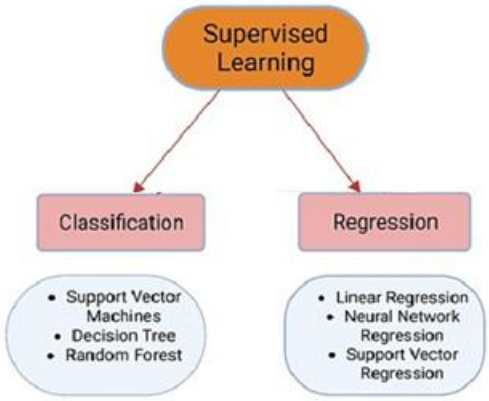

Figure 2.2: Supervised Machine Learning Categories [30]

2.2.2 Unsupervised machine learning

Unsupervised learning is a category of machine learning where models are trained on input data without any corresponding output labels. The goal is to identify inherent patterns, structures or groupings within the data. These algorithms are especially useful in exploratory data analysis when the structure of the data is not known in advance. The two most commonly used categories of unsupervised learning methods are clustering and dimensionality reduction.

- **Clustering**: These algorithms aim to group similar data points together based on defined similarity metrics. The goal of clustering is to identify patterns and relationships in the data without any prior knowledge of the data's meaning. It is mostly used in applications like market segmentation, social network analysis and recommendation systems.
- **Dimensionality reduction**: These algorithms try to reduce the number of input features while preserving the structure and important information in the data. These

are particularly valuable for visualization and noise reduction, especially when working with high-dimensional data.

- **Association rule learning**: Also known as association rule mining. It is a common technique used to discover associations. This is a rule-based ML technique that finds out some very useful relations between the parameters of a large dataset. It is used for market basket analysis that helps to better understand the relationship between the different products. The difference between association rule learning and clustering is that clustering groups similar data points based on inherent patterns, whereas association rule learning identifies meaningful relationships and co-occurrences among variables within a dataset.

2.2.3 Semi-supervised machine learning

Semi-supervised learning represents a learning method which unites supervised and unsupervised learning algorithms. The training process utilizes both a limited number of labeled examples and a substantial amount of unlabeled data. The approach proves valuable for real-world applications because it handles expensive or time-consuming labeling requirements of small datasets while utilizing abundant unlabeled data that is easily accessible.

These algorithms start by learning from labeled examples before using this knowledge to predict labels or patterns in unlabeled data. The integration of unlabeled data occurs through three main strategies which include self-training and co-training and graph-based methods.

- **Self-training**: Involves training a model on the labeled data and then using it to label the most confident examples from the unlabeled set, which are subsequently added to the training pool.
- **Co-training**: Trains multiple models on different feature subsets and allows them to label new data for one another.
- **Graph-based approaches**: Represent data as nodes in a graph, where label information is propagated to nearby unlabeled nodes based on similarity.

Semi-supervised learning has applications in areas such as natural language processing, image recognition, fraud detection and bioinformatics. These methods offer a practical solution in domains where labeled data is limited but the need of accurate models remain critical.

2.2.4 Reinforcement learning

Reinforcement learning enables agents to discover optimal decisions through environmental interactions by maximizing accumulated rewards. The learning method of reinforcement differs from supervised and semi-supervised learning because it does not need labeled input-output pairs. The agent develops its knowledge through experimentation by receiving feedback in the form of rewards or penalties after each action it performs.

The learning paradigm involves the agent perceiving environmental states to select actions from its policy before obtaining rewards and moving into new states. The agent seeks to discover an optimal policy which selects the most beneficial action for each state to obtain maximum long-term rewards. The approach excels at handling sequential decision-making challenges because it considers both present actions and past states and decisions when determining outcomes. The learning process uses Markov Decision Processes as a mathematical framework to describe agent-environment interactions. The fundamental components of reinforcement learning include:

- **Agent**: The learner/decision making.
- **Environment**: The external system the agent interacts with.
- **State**: A representation of the current situation of the environment.
- **Action**: A set of possible moves the agent can make.
- **Reward**: The feedback signal received after an action, indicating the desirability of the outcome.
- **Policy**: The strategy used by the agent to determine actions based on states.
- **Value Function**: An estimate of future rewards from a given state or state-action pair.

Popular reinforcement learning algorithms include Q-Learning, SARSA, and Deep Q-Networks (DQNs), as well as advanced methods like Proximal Policy Optimization (PPO) and Actor-Critic models. It has successfully applied in a variety of domains, such as robotics, autonomous vehicles, game playing, resource management and personalized recommendation systems.

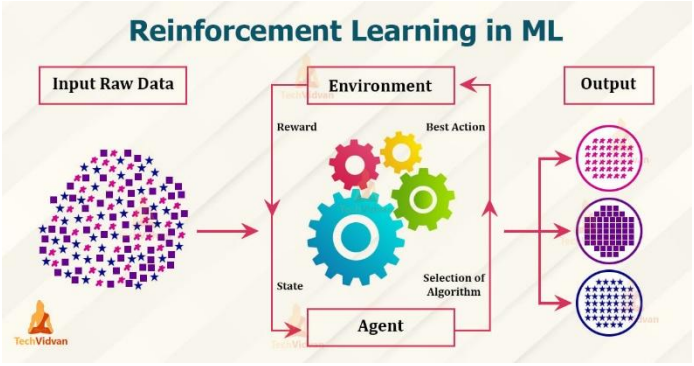

Figure 2.3: Reinforcement learning. [33]

Aspect	Supervised Learning	Unsupervised Learning	Semi-supervised Learning	Reinforcement Learning
Labeled Data	Required for all examples	Not used	Used partially (few labeled, many unlabeled)	No labeled input *(feedback via rewards)*
Goal	Learn a mapping from inputs to outputs	Discover hidden structure or patterns	Improve *learning with limited labels*	Learn optimal actions to maximize cumulative reward
Examples needed	Large, labeled *dataset*	Large unlabeled dataset	Small labeled + large unlabeled datasets	Repeated interactions with environment
Output	Predictions *(class or value)*	Groups, components *or features*	Predictions using learned representations	Policy for choosing actions
Common *algorithms*	Linear regression, SVM, Decision trees, Neural networks	K-means, PCA, Autoencoders	Self-training, Co-training, Graph-*based*	Q-learning, SARSA, DQN
Applications	Spam detection, fraud prediction, disease diagnosis	Market segmentation, anomaly detection	Medical image classification, speech recognition	Robotics, autonomous navigation
Challenges	Requires large labeled *datasets*	Lack of ground truth for evaluation	Risk of learning from incorrect pseudo-labels	Delayed rewards, high sample complexity

Table 2.1: Comparative overview of machine learning algorithms

2.3 LEARNING MEACHANISMS AND OPTIMIZATION IN MACHINE LEARNING

At the core of every machine learning model lies the process of training. Training is an iterative optimization where the model improves its ability to make accurate predictions by minimizing a predefined objective function based on the training data. Regardless of the learning type (Supervised, Unsupervised) this optimization process highlights how models generalize from data. This section introduces the fundamental mathematical principles that

are in control of how the machine learning models learn, focusing on optimization techniques, loss functions and training dynamics.

2.3.1 Objective functions and loss

Machine learning models are typically trained to minimize a specific function that quantifies how well the models' predictions align with the actual target outputs. This function is also known as loss function (or objective function) and its choice depends on the task (classification or regression in supervised learning or reconstruction in unsupervised models).

Loss function in supervised learning

For supervised tasks like classification, the model predicts discrete categories. One common loss function in binary classification is the binary cross-entropy loss defined as:

$$L_{BCE}(y, \hat{y}) = [ylog(\hat{y}) + (1 - y)\log(1 - \hat{y})]$$

Where:

- y ∈ {0,1} is the training label
- $\hat{y}$ ∈ (0,1) is the predicted probability

For multiclass classification tasks (such as the anomaly detector discussed later in this thesis), categorial cross-entropy is commonly used as the loss function. It extends binary cross-entropy to handle problems involving more than two classes by comparing the true class distribution with the predicted probability distribution across all classes. The categorial cross-entropy is defined as this:

$$L_{CCE}(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

Where:

- C is the number of possible classes
- y is one-hot encoded vector representing the true class
- $\hat{y}$ is the predicted probability vector across all classes

For regression problems, where outputs are continues values, the mean squared error is mostly used:

$$L_{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where:

- n is the total number of samples
- y_i is the true target value for the i-th sample
- $\hat{y}_i$ is the predicted value for the i-th sample

This function penalizes the squared difference between actual and predicted values, making it particularly sensitive to large errors.

Loss in unsupervised learning

In unsupervised learning the loss function often measures how well the model reconstructs the input data. The mean squared error is also used here, where the model learns to compress and then reconstruct inputs:

$$L_{reconstruction}(x, \hat{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$$

Where:

- x is the input
- $\hat{x}$ is the reconstructed input

2.3.2 Gradient-based optimization

Most machine learning models are optimized using gradient based methods, where the loss function is minimized by updating the model parameters in the direction of the steepest descend. The most used algorithm is gradient descent.

Gradient descent updates model parameters θ iteratively using:

$$\theta = \theta - \eta \nabla_\theta L(\theta)$$

Where:

- θ is the parameters of the model
- η is the learning rate, a small positive scaler controlling the step size

- $\nabla_\theta L(\theta)$ is the gradient (vector of partial derivatives) of the loss function with respect to model parameters.

This process continues until convergence (while the gradient becomes very small or the loss stops improving).

Advanced optimization algorithms

To improve convergence and reduce sensitivity to learning rate selection, several improved optimizers are used:

- **Momentum**: Accelerates gradient descent in relevant directions.
- **RMSprop**: Adapts learning rate per parameter using a moving average of squared gradients.
- **Adam (Adaptive moment estimation)**: Combines momentum with RMSprop. Widely used for training deep neural networks.

2.3.3 Regularization and generalization

While minimizing loss on the training data is important, doing so too aggressively can lead to overfitting. To combat this, regularization techniques are applied to penalize overly complex models. It introduces an additional penalty term into the loss function to discourage excessive model complexity, such as overly large weights (see APPENDIX A: UNDERSTANDING WEIGHTS IN MACHINE LEARNING) or overly flexible decision boundaries (see APPENDIX B: DECISION BOUNDARIES IN MACHINE LEARNING).

Two common regularization terms are:

L1 regularization

$$\Omega(\theta) = \lambda \sum_i |\theta|$$

Encourages sparsity in the model by driving many weights to exactly zero. Leads to simpler models and can be used for feature selection, as irrelevant features tend to be assigned to zero weight. It is used in high-dimensional datasets where only a subset of features is informative.

L2 regularization

$$\Omega(\theta) = \lambda \sum_i \theta^2$$

Encourages the model to keep weights small but not necessarily zero. Helps to prevent the model from relying too heavily on a single input feature and improves numerical stability. It is used in most common ML models, such as logistic regression, neural networks and SVMs.

In both cases, the regularization term Ω(θ) is scaled by a hyperparameter λ, which controls the strength of penalty. A larger λ imposes stronger regularization, encouraging simpler models. A smaller λ allows the model to focus more on minimizing the original data loss.

Total Objective Function

The modified loss function, which includes both the data loss and regularization term becomes:

$$L_{total} = L_{data} + \Omega(\theta)$$

By incorporating regularization into the training process, machine learning models are better equipped to generalize (to maintain strong predictive performance) when applied to new, unseen data. This is particularly critical in real-world scenarios like cybersecurity, where models must adapt to constantly changing patterns and behaviors.

2.4 EVALUATION METRICS FOR DETECTION SYSTEMS

The evaluation of machine learning models, particularly those applied in detection systems such as intrusion detection or anomaly recognition, requires a subtle set of performance metrics. Relying solely on simple measures like accuracy may yield misleading conclusions. This is most possible to happen in imbalanced datasets where anomalies are rare (common in cybersecurity anomaly detection). As a result, a range of statistical metrics based on confusion matrixes are commonly used to assess model quality with respect to both detection capacity and error behavior.

2.4.1 Confusion matrix and core definitions

The confusion matrix is a fundamental tool in classification analysis. It summarizes the outcomes of predictions made by the model compared to the actual labels:

	Predicted positive	**Predicted negative**
Actual positive	True positive (TP)	False Negative (FN)

	Predicted positive	Predicted negative
Actual negative	False Positive (FP)	True Negative (TN)

Table 2.2: Confusion matrix for binary classification.

- **True positive (TP)**: The model correctly identified a positive case (for example an actual intrusion).
- **True negative (TN)**: The model correctly identified a normal (benign) case.
- **False positive (FP)**: A benign input incorrectly flagged as anomaly (false alarm).
- **False negative (FN)**: A malicious input went undetected by the model (missed detection).

These values are the basis to evaluate the following metrics

Accuracy

Accuracy represents the proportion of correctly classified instances. Although commonly reported, it can be misleading in imbalanced datasets where the majority class dominates. A high accuracy may simply reflect the model's ability to correctly recognize only the majority class while ignoring the minority one.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision

Precision evaluates how many of the positive predictions were correct. High precision indicates a low false alarm rate, which is particularly important where false positives are costly.

$$Precision = \frac{TP}{TP + FP}$$

Recall

Recall (sensitivity / true positive rate) measures the ability of a model to detect true positives. It is crucial in applications, where failing to identify a positive case is far more dangerous than raising a false alarm. (Crucial in anomaly detection)

$$Recall = \frac{TP}{TP + FN}$$

F1 score

The F1 score is the harmonic mean of precision and recall. It provides a balanced metric that is especially useful in imbalanced datasets where neither precision nor recall alone gives a complete picture of performance.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Multiclass Classification: Macro and Micro Averaging

In binary classification, performance metrics like precision, recall and f1 score are computed using a single set of true positives, false positives, true negatives and false negatives. However, in multiclass classification problems, these metrics must be adapted to account for multiple classes.

Two widely used strategies to aggregate performance across classes are macro-averaging and micro-averaging. Each provides a different perspective on model performance, especially in the presence of class imbalance, which is common in real-world cybersecurity datasets.

Macro averaging computes the evaluation metric independently for each class and then averages the results:

$$Macro\ F1 = \frac{1}{C}\sum_{i=1}^{C} F1_i$$

Where:

- C is the number of classes
- $F1_i$ is the F1 score of the i-th class

Macro averaging treats all classes equally, regardless of their frequently in the dataset. By computing evaluation metrics independently for each class and then averaging the results, this approach ensures that minority classes are not overshadowed by the performance on dominant classes. As a result, it is particularly useful for highlighting underperformance in rare or less frequent categories.

On the other hand, micro averaging aggregates the contributions of all classes to compute a global metric.

$$Micro\ Precision = \frac{\sum TP_i}{\sum TP_i + \sum FP_i}, \qquad Micro\ Recall = \frac{\sum TP_i}{\sum TP_i + \sum FN_i}$$

Micro averaging gives more weight to frequent classes by aggregating true positives, false positives and false negatives across all classes before computing the evaluation metric. This approach reflects the overall instance-level performance of the model rather than evaluating each class individually. It is particularly useful in applications where the total number of correct predictions is more important than balanced class-level performance. However, it may obscure poor performance on minority classes, as the contributions of dominant classes can overshadow errors made in rare but potentially critical categories.

Threshold Selection

The output of many classification problems including probabilistic models like logistic regression and neural networks produces probabilities instead of definite labels. The conversion of probabilities into class predictions requires the application of a decision threshold. The threshold value can be modified to achieve the right balance between recall and precision according to the specific application requirements.

The output of unsupervised learning models including autoencoders consists of reconstruction errors for each input instead of class probabilities. A threshold value must be established for error measurements to identify points as anomalous. The selection of this threshold depends on validation data or percentile-based methods. The selection of proper thresholds remains essential for both supervised and unsupervised methods to achieve the optimal balance between false positives and false negatives.

2.5 ANOMALY DETECTION IN CYBERSECURITY

Anomaly detection is a critical technique in the field of cybersecurity. It aims at identifying data points, patterns or behaviors that significantly deviate from what is considered normal. In security context, normal is typically defined based on legitimate system behavior, user activity or traffic patterns. Any deviation from this learned or predefined norm may indicate the presence of a security threat such as intrusion, misconfiguration or insider attack.

2.5.1 Signature-Based vs Anomaly-Based Detection

There are two primary types of intrusion detection. Signature-based and anomaly-based. Signature-based detection relies on predefined rules and databases of known attack patterns to identify malicious activity. This method is closely related to supervised learning, as it requires labeled examples of both normal and malicious behavior in order to train a classified to recognize known threats. While signature-based detection systems are

accurate for detecting previously observed attacks, they are ineffective against zero-day attacks [3] or evolving threats.

Anomaly-based detection systems identify abnormal behavior patterns without needing prior attack knowledge. The method corresponds to unsupervised learning because models receive only normal data training to detect major deviations as potential anomalies. Anomaly-based systems excel at detecting zero-day attacks and insider threats and complex attack strategies that do not match existing signatures. The detection method generates additional false positive results because unusual behavior does not always indicate malicious activity.

This thesis adopts a hybrid approach that leverages both unsupervised and supervised learning. Autoencoders are used for unsupervised anomaly detection, while a supervised model is used to classify known attacks, enabling both adaptability and accurate threat categorization.

2.6 AUTOENCODERS

Autoencoders are a class of artificial neural networks used primarily for unsupervised learning, particularly in tasks such as dimensionality reduction, representation learning and anomaly detection. Their core functionality lies in learning efficient encodings of input data by compressing and reconstructing it through a bottleneck structure. In cybersecurity, autoencoders have gained popularity for modeling normal behavior in network traffic and identifying deviations indicative of potential intrusions or attacks.

2.6.1 Structure

An autoencoder consists of two main components: an encoder and a decoder. The encoder transforms the input data into a lower-dimensional representation, also known as the latent space or bottleneck layer. The decoder then attempts to reconstruct the original input from this compressed representation.

Formally, for an input vector z $\in R^n$, the encoder is a function f_0 that maps the input to a latent representation z $\in R^m$, where m < n

$$z = f_0(x)$$

The decoder, g_f maps z back to a reconstruction $\hat{x}$:

$$\hat{x} = g_f(z)$$

Both f_0 and g_f are typically implemented as fully connected neural networks, although convolutional or recurrent variants exist for specialized data types. Mathematical formulation

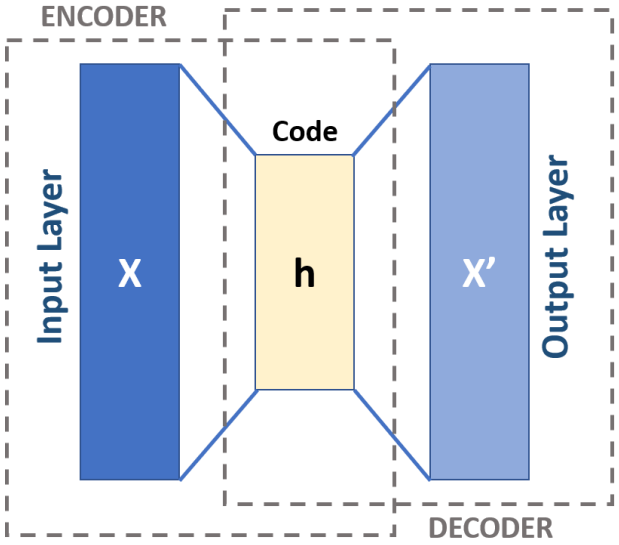

Figure 2.4: Autoencoder structure. [34]

2.6.2 Mathematical formulation

The objective of training an autoencoder is to minimize the difference between the input x and its reconstruction $\hat{x}$. This is typically formalized as the minimization of a loss function over a dataset X = $\{x^{(1)}, x^{(2)}, \dots, x^{(N)}\}$:

$$L_{(\theta,\varphi)} = \frac{1}{N}\sum_{i=1}^{N} ||x^{(i)} - g_f(f_0(x^{(i)}))||^2$$

This loss represents the average reconstruction error, commonly measured using Mean Squared Error (MSE), although other metrics such as Mean Absolute Error (MAE) can also be used depending on the application.

2.6.3 Reconstruction loss

The reconstruction loss quantifies how well the model can reproduce the original input. n anomaly detection, the assumption is that the autoencoder, trained solely on normal data, will reconstruct normal samples accurately (low loss), while it will struggle to reconstruct abnormal or anomalous samples (high loss). This difference in reconstruction loss can be

used as a basis for detection (low loss equals likely normal, while high loss means that the input is probably anomalous).

To distinguish between normal and anomalous instances, a threshold is applied to the reconstruction error produced by the autoencoder. A fixed threshold can be set using a high percentile (e.g., 95th or 99th) of reconstruction errors from validation data, under the assumption that normal behavior will yield low error. In this thesis, a dynamic thresholding approach is also employed, where the threshold adjusts over time based on recent error distributions. This allows the system to adapt to changes in network behavior and reduces false positives caused by benign but evolving traffic patterns.

2.6.4 Use in detecting anomalies

Autoencoders are particularly well-suited for unsupervised anomaly detection because they do not require labeled data and are capable of modeling complex data distributions. In the context of network intrusion detection, they are trained on benign network traffic to learn the typical structure and behavior of normal flows. When exposed to anomalous traffic, the autoencoder fails to reconstruct them accurately, producing a higher reconstruction error.

This behavior makes autoencoders effective at detecting zero-day attacks, insider threats and subtle deviations in user behavior. Their lightweight architecture and generalization capabilities also make them suitable for integration in real-time systems.

CHAPTER 3: RELATED WORK

The scientific basis of this thesis appears in this chapter. The research examines previous studies about machine learning applications in network intrusion detection systems (NIDS) especially those using autoencoders and hybrid architectures and real-time traffic analysis. The research findings presented in this section directly shaped the system architecture and training methods and evaluation procedures of the proposed system in this thesis.

3.1 AUTOENCODER BASED INTRUSION DETECTION

The established concept of using autoencoders for cybersecurity anomaly detection exists but practical deployments remain below signature-based systems. The research paper Kitsune [11] presented an ensemble of lightweight autoencoders which learned to monitor traffic feature subsets. The research showed that using multiple small autoencoders which cover different parts of the input space produces better generalization and minimizes the risk of overfitting. The system architecture supports real-time online anomaly detection that identifies zero-day threats effectively in network environments.

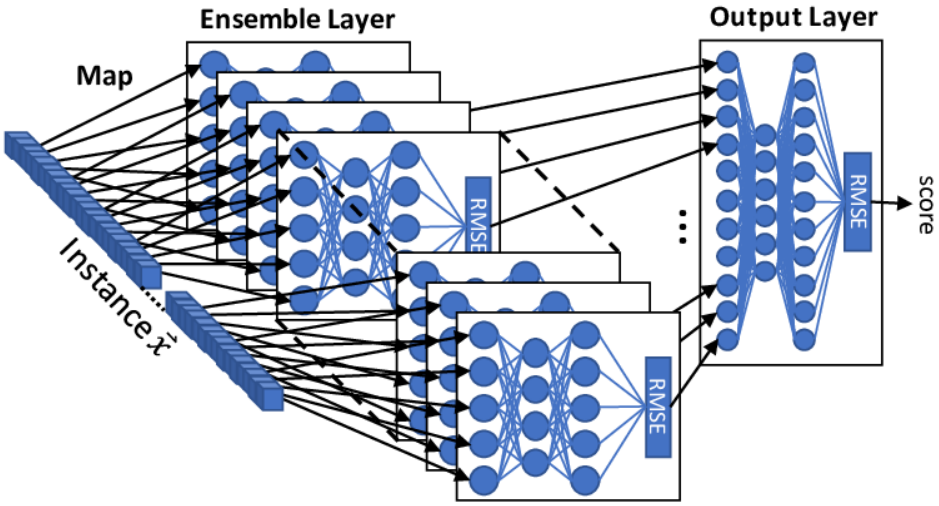

Figure 3.1: Kitsune ensemble of autoencoders architecture. [11]

The figure above illustrates the architecture of Kitsune's autoencoder ensemble. The input instances undergo a transformation into smaller feature subsets which are processed independently by multiple shallow autoencoders in the ensemble layer. The autoencoders specialize in different parts of the input space because they learn to reconstruct only the observed features. The autoencoders operate simultaneously to generate reconstruction errors which are usually calculated through Root Mean Squared Error (RMSE). The output layer receives aggregated errors from all sub-models before producing a final anomaly score through another neural network. The layered architecture enhances generalization

capabilities and minimizes overfitting risks to specific feature groups while maintaining real-time detection efficiency through lightweight and modular autoencoders. The design allows Kitsune to handle increasing feature dimensions while maintaining fast performance in high-speed environments.

The research paper AnomalyDAE [12] developed a dual autoencoder system to identify anomalies within attributed networks. The AnomalyDAE system differs from Kitsune because it uses two autoencoders to model structural data and node attributes separately. The dual-layered data representation leads to better detection accuracy because it enables the system to learn more complex data patterns. The method shows superior performance in minimizing false positives while maintaining scalability for handling big data sets.

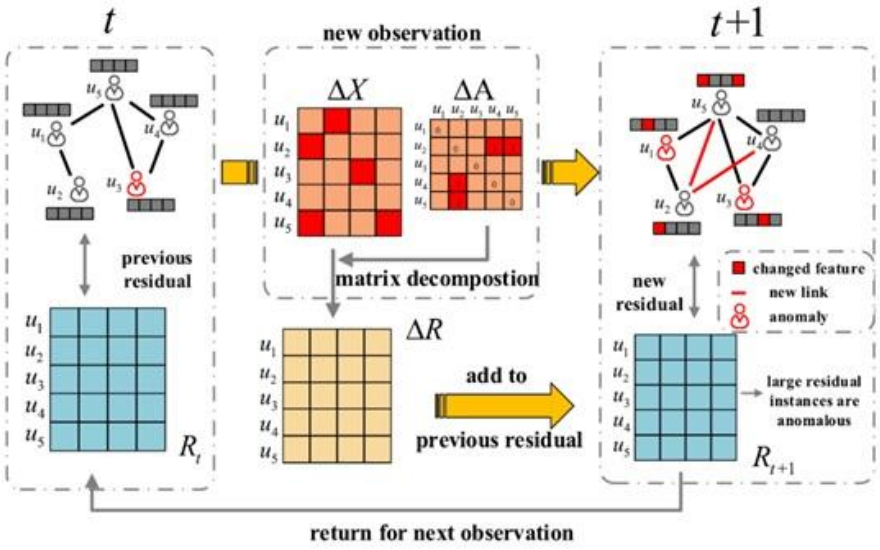

Figure 3.2: AnomalyDAE architecture. [12]

The AnomalyDAE framework shows its architectural pipeline through this figure. The model processes dynamic attributed networks through simultaneous analysis of graph topology changes and node feature variations across consecutive time windows. The model generates the residual matrix R_t at each time step t. The model updates the residual matrix through matrix decomposition after receiving new observations that include changes in node attributes (ΔX) and connectivity (ΔA). The new residual matrix R_{t+1} results from adding deltas (ΔR) to the previous residual. The model identifies nodes or substructures with significant residuals as anomalous. The model architecture allows it to monitor temporal patterns and detect small changes in structure or attributes while reducing false alarm rates. AnomalyDAE achieves better precision in complex time-varying network environments through its dual autoencoders which learn structure and attribute data independently.

The research findings led to the creation of fundamental autoencoder and Dual Autoencoder implementations in this research. The AnomalyDAE framework reproduction was limited by resource constraints because of missing adjacency matrix generation capabilities but the system implemented a simplified version of structural and attribute feature combination.

3.2 HYBRID MACHINE LEARNING APPROACHES FOR IDS

Hybrid machine learning architectures represent a promising solution for intrusion detection since they work well in environments with fast-evolving attacks and limited labeled data. These systems leverage the adaptive anomaly detection abilities of unsupervised autoencoders alongside the classification strengths and interpretability features of supervised LightGBM models. Systems employing this dual-layered approach can detect both familiar and unknown threats with flexibility and scalability.

The paper "A two-stage intrusion detection method based on light gradient boosting machine and autoencoder" [13] presents a two-stage intrusion detection framework which integrates autoencoders with LightGBM classifiers. The method uses recursive feature elimination (RFE) for feature selection followed by focal loss function application to enhance minority attack class sensitivity. A dual prediction process which combines LightGBM with autoencoders operates as the final stage to enhance accuracy and preserve resistance to novel attacks.

"Hybrid Neural Network-Based Intrusion Detection System: Leveraging LightGBM and MobileNetV2 for IoT Security" [14] developed a hybrid neural intrusion detection system which uses LightGBM for analyzing network flow data and MobileNetV2 for analyzing packet data. The approach provides high detection precision along with computational efficiency that makes it appropriate for IoT systems and resource-constrained networks.

The research "AutoEncoder and LightGBM for Credit Card Fraud Detection Problems" [15] presented an autoencoder-based detection system with LightGBM classification that initially targeted fraud detection but works well for cybersecurity applications.

Research indicates that using unsupervised learning for early anomaly detection alongside supervised learning for detailed threat classification produces successful results. The proposed system in this thesis functions through autoencoders for real-time traffic anomaly detection alongside LightGBM classification for flows with labeled data. This system design satisfies operational deployment requirements through its balanced detection accuracy and model interpretability and runtime efficiency.

The thesis implements an autoencoder-based anomaly detection system that works in conjunction with a LightGBM classifier. While prior research has shown that hybrid

models combining autoencoders and classifiers like LightGBM can improve detection accuracy, most of these systems are evaluated only in offline scenarios using preprocessed datasets. In contrast, this thesis contributes a lightweight, real-time framework capable of live traffic capture, anomaly detection, attack classification, and immediate defensive response via IP blocking. Its modular architecture, practical deployment readiness, and real-time response capabilities set it apart from existing academic approaches.

3.3 COMPARISON WITH EXISTING WORK

Study	Approach	Dataset	Real-time	Output
Kitsune	Ensemble of autoencoders	Mirai + benign	Yes	Anomaly score
AnomalyDAE	Dual Autoencoder	Attributed graphs	No	Node-level anomalies
"A two-stage intrusion detection method based on LightGBM and autoencoder"	LightGBM + Autoencoder (two-stage)	UNSW-NB15	No	Attack Type
This thesis	Autoencoder + LightGBM + real-time blocking	UNSW-NB15 + Kyoto	Yes	Detection + Attack Type + IP blocking

Table 3.1: Comparison between thesis and related work.

The evaluation of this thesis requires an examination of its methodology against existing models. Table 3.1 presents the main distinctions between this system and the most relevant autoencoder-based or hybrid IDS models described in this chapter.

The evaluation process assesses each study based on its methodological structure and dataset selection and real-time capabilities and final output. The real-time architecture of Kitsune [11] stands out through its ensemble of autoencoders for anomaly scoring yet lacks both semantic classification and active response capabilities. The dual autoencoder design in AnomalyDAE [12] detects graph-based anomalies yet it operates on attributed graphs and lacks real-time processing capabilities. Anomaly DAE developed a two-stage process that combines LightGBM with autoencoders for offline classification purposes. The thesis

presents a complete hybrid system which detects anomalies in real-time through autoencoders while using LightGBM for threat classification and executes immediate IP blocking responses. The proposed approach unites detection with classification and defensive response in a single deployable system to connect research models with operational network defense capabilities

CHAPTER 4: DATASETS AND FEATURE SELECTION

A machine learning-based intrusion detection system requires high-quality training and evaluation data together with relevant and informative features that represent network traffic behavior to achieve reliable performance. The development of cybersecurity models requires two essential components: representative datasets and robust feature engineering to achieve generalization to new threats and real-time operation and environmental adaptation.

This thesis employs the UNSW-NB15 and Kyoto 2006+ [16], [17] datasets which are explained in detail throughout this chapter. The datasets differ from each other regarding their structural organization and labeling systems and realistic nature. The UNSW-NB15 dataset provides extensive labeled network traffic data for supervised learning applications yet Kyoto 2006+ delivers extended real-world traffic data for both anomaly detection and model generalization evaluation.

The following section explains the feature engineering process. The conversion of raw packet or flow data into numerical structures happens through flow-based features which include duration, packet count, byte volume and protocol metadata. The selection of each feature occurs because it detects behavioral anomalies in traffic and researchers support its choice through domain knowledge and existing research. The chapter explains essential preprocessing operations including label encoding and normalization and class balancing methods which are necessary for successful training of supervised and unsupervised models.

4.1 DATASETS OVERVIEW

The two publicly available benchmark datasets UNSW-NB15 and Kyoto 2006+ served to train and evaluate the hybrid intrusion detection system described in this thesis. Two intrusion detection research datasets were chosen because they contain attack scenarios and provide labeled attack information through UNSW-NB15 while Kyoto offers unlabeled real-world traffic behavior.

The Australian Centre for Cyber Security (ACCS) developed the UNSW-NB15 dataset through its creation of attack simulation environments which use contemporary tools to capture modern attack types. The dataset contains more than 2.5 million flow records that feature 49 extracted elements obtained from Argus and Bro-IDS tools. The dataset contains two types of labels because it includes binary normal vs. attack classification together with multiclass attack classification into ten attack categories including DoS, Exploits, Fuzzers, Backdoors, Shellcode, Worms, and more. The labeled flows serve as optimal training data for LightGBM alongside other supervised models because they provide distinct class

boundaries between benign and malicious behavior. The UNSW-NB15 dataset features flow-based detection features which match the system's detection pipeline requirements.

The Kyoto 2006+ dataset contains network traffic data from honeypot systems that span more than ten years. Each flow record contains more than 24 features which include TTL information as well as packet volume and flow direction and duration and connection state metrics. Autoencoders benefit from the attack class label deficiency in UNSW-NB15 because Kyoto lacks explicit labeling in many of its versions. The dataset provides useful information for testing the system's anomaly detection capabilities when operating normally because of its continuous nature and authentic network behaviors. The real-world traffic patterns in this dataset help prevent the model from learning behaviors that only exist in artificial simulation environments.

The thesis combines UNSW-NB15 labeling with Kyoto 2006+ real-world traffic patterns to evaluate the proposed system using both controlled attack scenarios and unlabeled traffic patterns which resemble real-world deployment. The use of two datasets provides enhanced validation of detection models in experimental and realistic environments.

4.2 UNSW-NB15 DATASET

The UNSW-NB15 dataset has become one of the most widely accepted and utilized resources in intrusion detection system (IDS) research due to its modern attack diversity, structured flow representation, and compatibility with both supervised and unsupervised machine learning paradigms. The dataset was created in response to the limitations of legacy datasets such as KDD'99, which lacked contemporary attack techniques and contained numerous redundant or unrealistic entries.

The database contains network flow records which represent summarized connections through the 5-tuple (source IP, destination IP, source port, destination port, protocol). The dataset contains more than 2.5 million flow records which receive either binary attack or normal labels or multiple multiclass attack category labels. The records exist in tabular format with 49 features that packet inspection tools extracted to provide detailed behavioral and structural attributes for each flow.

What distinguishes UNSW-NB15 from earlier benchmarks is not only the inclusion of multiple modern attack types but also the quality and diversity of its features. These include:

- Basic statistical features (e.g., packet and byte counts)
- Time-based metrics (e.g., flow duration, rate of transmission)
- Header-related values (e.g., TTL, TCP flags)

- Content inspection fields (e.g., state, service, anomaly score)

This broad set of features supports both high-level behavior modeling and fine-grained attack pattern recognition, aligning closely with the needs of flow-based detection pipelines such as the one proposed in this thesis.

The dataset provides advantages but requires preprocessing to solve its technical issues:

- Neural network training requires normalization of features because they show skewed distributions.
- The model requires label encoding or one-hot encoding for categorical features (e.g., protocol, state) based on the selected model type.
- The multiclass labels show a significant class imbalance because some attack categories appear less frequently than others. The class imbalance problem in supervised models requires attention because it leads to performance degradation unless addressed through stratified sampling or SMOTE-based synthetic data augmentation techniques.

The structured design and clear labeling of UNSW-NB15 makes it suitable for training supervised classifiers like LightGBM and benchmarking unsupervised anomaly detectors like autoencoders when using only benign flows. The dataset structure allows task-dependent sampling and preprocessing which makes it an excellent resource for building hybrid detection systems.

4.3 KYOTO 2006+ DATASET

The Kyoto 2006+ dataset is recognized as one of the most valuable long-term resources for evaluating intrusion detection systems in realistic, production-like conditions. It was developed at Kyoto University by collecting data from a group of low-interaction honeypots that passively recorded inbound and outbound traffic over the course of several years. Unlike synthetic datasets generated in controlled lab environments, Kyoto 2006+ captures traffic as it occurs in practice, including typical background noise, benign connections, and unsolicited traffic from external hosts attempting to probe or exploit the network.

Each record in the dataset represents a network connection or flow, described using more than 20 metadata features. These include not only traditional traffic descriptors such as IP addresses, ports, and protocols, but also behavioral attributes such as:

- Packet and byte volume
- Time-to-Live (TTL) values
- Connection states (e.g., SYN_SENT, ESTABLISHED, CLOSED)

- Service types and flags
- Inbound/outbound ratios

Because the dataset is collected continuously over time, it allows for the modeling of traffic evolution, cyclical behavior, and long-term trends, which are critical when evaluating anomaly detection algorithms in non-stationary environments. It also introduces complexities that are often absent in simulation-based datasets, such as:

- Irregular traffic bursts
- Incomplete or malformed connections
- Protocol tunneling or port misuse
- Overlapping flows and long idle durations

From a research perspective, Kyoto 2006+ is particularly useful for training and testing unsupervised learning algorithms, where the objective is to identify deviations from observed baseline behavior. In this thesis, it is leveraged as a testbed for evaluating the generalization performance of autoencoder-based anomaly detection, especially under unlabeled conditions where no prior attack annotations are available.

One of the distinguishing characteristics of the dataset is its temporal and operational continuity. Rather than generating discrete batches of attacks and normal traffic, Kyoto logs all activity as it naturally occurs. This enables models to capture subtle shifts in network behavior, which may not immediately manifest as attacks but could indicate reconnaissance, misconfiguration, or emerging threats.

While the dataset lacks explicit attack class labels in most of its available forms, this limitation is offset by its representative and heterogeneous traffic, which challenges models to learn from noise, variability, and unbalanced event distributions. As a result, it plays a key role in validating the robustness of the unsupervised detection pipeline presented in this thesis, complementing the structured, labeled design of UNSW-NB15.

4.4 FEATURE SELECTION

Effective anomaly detection in cybersecurity requires a basic understanding of how network traffic is structured. Network communication is carried out through small units of data called packets, each containing a header and payload. The header includes metadata such as source and destination IP addresses, port numbers and packet size.

Common transport layers include:

- **TCP (Transmission Control Protocol)**: Reliable, connection-oriented communication.
- **UDP (User Datagram Protocol)**: Faster, connectionless but less reliable.

A sequence of packets between two endpoints forms a flow, identified by the 5-tuple: source IP, destination IP, source port, destination port and protocol. Flow-based intrusion detection features are extracted from these flows, such as:

- **Duration**: How long the flow lasted
- **Packet counts**: Total number of packets sent/received
- **Byte volume**: Amount of data transferred
- **TTL (Time to Live)**: Number of network hops remaining (see APPENDIX C: UNDERSTANDING NETWORK HOPS)
- **Protocol**: TCP, UDC

The mentioned features serve as essential indicators of traffic anomalies because they demonstrate consistent communication patterns which remain stable during normal operations but change dramatically during attacks. For example, a flooding attack becomes evident through sudden increases in flow duration and packet count while scanning attempts reveal themselves through sequential port connections from the same IP address. The detection of misconfigured or malicious behavior becomes possible through the identification of abnormal TTL values and unexpected protocol usage. Anomaly detection systems use low-level traffic characteristic analysis to identify minor deviations which traditional rule-based methods would miss.

Feature engineering is a crucial step in the development of any machine learning-based intrusion detection system, as the selected input features determine both the quality of the model's learning and its ability to generalize to unseen behavior. In this thesis, the goal was to identify a subset of flow-level features that are both computationally efficient and highly expressive in distinguishing normal network activity from malicious or anomalous behavior. Feature selection was based on domain knowledge, dataset exploration, and compatibility with both the autoencoder (unsupervised anomaly detection) and LightGBM (supervised classification).

The chosen features reflect a balance between statistical traffic patterns, protocol-level characteristics, and behavioral attributes. These features were extracted from the raw datasets (UNSW-NB15 and Kyoto 2006+) and preprocessed to remove redundant, irrelevant, or low-variance fields. In total, 11 core features were retained for modeling:

Feature	Description
dur	Duration of the flow (seconds)
proto	Transport-layer protocol
state	Connection state
spkts	Packets sent by the source
dpkts	Packets received by the destination
sbytes	Bytes sent from source to destination
dbytes	Bytes sent from destination to source
sload	Average bits/second from source to destination
sttl	Time to live (TTL) value from source
dttl	TTL value from destination
dload	Average bits/second from destination to source

Table 4.1: Selected features from datasets.

The selected features work in encrypted environments and payload inspection restrictions because they do not depend on packet content. The features work well for real-time processing in production environments because they enable low-latency detection.

The preprocessing phase included label encoding categorical features `proto` and `state` to make them suitable for machine learning models. The continuous features received min-max scaling normalization to create values between 0 and 1 which benefits neural network training and autoencoder reconstruction error stability.

The analysis of feature correlations helped identify redundant features which were then removed through manual inspection to eliminate non-informative fields including timestamps and identifiers and constant-value columns. Selected attributes enhanced,

detection accuracy, and reduced computational cost, comparing with using every one of the features.

These features create a behavior-based input representation which enables detection of multiple attack types that will be studied in detail during the following analysis of attack categories and behavioral characteristics.

4.5 ATTACK CATEGORIES AND BEHAVIORAL CHARACTERISTICS

Intrusion Detection Systems (IDS) rely not only on statistical patterns in traffic features, but also on understanding how different types of cyberattacks manifest distinct behavioral anomalies in network flows. Each attack type affects the network in specific ways, from sudden spikes in traffic to abnormal flow directionality, which can be detected by analyzing engineered features. This section explores the most common network attack categories found in benchmark datasets, particularly those in UNSW-NB15, and explains how each type typically alters traffic behavior, enabling detection via anomaly-based or signature-based techniques.

4.5.1 Denial-of-Service (DoS) and Distributed DoS (DDoS)

The main goal of DoS attacks is to make a service or system unavailable through excessive traffic flooding. The service becomes unavailable to legitimate users because the attacks consume all available bandwidth memory and CPU resources. The attacks originate from a single source.

The distributed nature of DDoS attacks makes them more powerful than DoS attacks because they use multiple infected machines (botnets) to carry out the attacks. The traffic patterns of DoS and DDoS attacks share the following characteristics:

- High connection frequency and packet count
- Long flow durations
- Unusually high or bursty byte volumes
- The same IP addresses or multiple IP addresses repeatedly attempt connections.

These attacks generate fast-growing packet counts and both high and uniform flow durations and large repetitive byte volumes that target a single system. The simultaneous patterns from multiple sources in DDoS attacks produce noticeable anomalies which can be detected through traffic behavior analysis. They are detectable through flow features such as `spkts`, `dpkts`, `dur`, and `sbytes`.

4.5.2 Exploits

Exploit attacks attempt to take advantage of software vulnerabilities in operating systems or applications. These attacks often generate traffic patterns that are context-specific but tend to include:

- Unusual payload structures or lengths
- Unexpected protocol usage
- Atypical sequence of request/response behavior

Software vulnerabilities serve as targets for exploitation attempts through the creation of payloads and the triggering of unintended system behavior. The detection of these attacks in flow-based features occurs through identifying abnormal patterns between protocol (`proto`), source port (`srcport`) and destination port (`dstport`) that differ from typical traffic behavior. The exploitation of flows results in abnormal state transitions because attackers use methods to skip standard connection establishment protocols. Anomaly detection systems identify these irregularities because they differ from standard application-layer behaviors thus marking them as potential exploit attempts.

4.5.3 Fuzzers

Fuzzing involves sending malformed or random input data to a program in an attempt to find vulnerabilities or trigger crashes. Fuzzers usually operate at high frequency, which may result in:

- Short flow durations (`dur`)
- Low `sbytes` or irregular `sload`
- A large number of TCP reset (RST) flags due to application crashes

Fuzzers may also generate a high number of failed connection attempts visible through specific state combinations and sequence anomalies.

4.5.4 Backdoors

System backdoor attacks require unauthorized access point installation to enable remote control and future access without detection. The attacks can stay dormant for extended periods before generating unexpected traffic surges. The following characteristics are typical of these attacks:

- Unusual traffic patterns emerge from atypical ports which include both high-numbered and non-standard ports.

- The traffic flows in both directions but the byte volume between the two directions remains unequal.
- The same endpoints maintain continuous connections with each other.

Backdoor attacks establish unauthorized and often covert communication channels that allow attackers to maintain persistent access to a compromised system. These communications typically exhibit irregular but sustained flow characteristics. Flow features such as destination bytes (`dbytes`) and destination packets (`dpkts`) can indicate abnormal volumes of data being sent from internal hosts to unfamiliar external destinations. Additionally, features like `ct_srv_src` (count of connections to the same service from the same source) and `ct_dst_ltm` (count of connections to the same destination in a time window) reveal repetitive access patterns that are uncommon in benign traffic. These indicators collectively help anomaly detection systems identify stealthy backdoor operations.

4.5.5 Shellcode

Shellcode attacks inject binary code into system memory, often exploiting buffer overflow vulnerabilities. These attacks are typically small in packet size but may trigger secondary flows once execution succeeds. Detection may rely on:

- Very small `sbytes` or `dbytes`
- Short duration followed by a burst of new flows from compromised endpoints

Although difficult to detect via traditional statistical models, deep learning models like autoencoders can flag the rare, nonlinear flow patterns shellcode may produce.

4.5.6 Generic Attacks

The broad category includes brute-force methods that attack authentication systems and encryption protocols and hash algorithms. These attacks exhibit:

- Repeated attempts over short timeframes
- Constant data sizes
- Similar port/protocol combinations repeated across sessions

The traffic patterns of generic attacks including brute-force and hash collision-based exploits show repetitive and aggressive behavior when targeting multiple destinations or services. The `ct_srv_dst` feature counts service and destination connections while `ct_state_ttl` combines connection states with TTL values and high connection count indicators serve to detect these behaviors. The attacks produce a large number of quick

automated attempts which result in steady TTL values and repeated service targeting statistical analysis and anomaly detection systems can identify.

4.5.7 Reconnaissance and Analysis

Reconnaissance (or scanning) is often the precursor to more intrusive actions. It involves port scanning, IP sweeps, and other probing techniques to gather network intelligence. Recognizable traits include:

- Large numbers of short-lived flows
- Flows that target many destination ports (`dport`) or IPs (`dstip`) in a short period
- Low packet count with TCP SYN flags only

The main goal of reconnaissance and analysis attacks involves systematic probing to gather information about potential targets through port and host scanning. The `ct_dst_ltm` (number of connections to the same destination host in a time window), `ct_srv_dst` (connections to the same service/destination), and `ct_src_dport_ltm` (connections from the same source IP to different destination ports) features effectively detect these behaviors. The scanning activity's breadth and frequency become apparent through these features which enable anomaly detection systems to detect potential reconnaissance efforts before an actual intrusion takes place.

4.5.8 Worms

Worms are self-propagating malware that replicate across networks without human intervention. They often result in:

- High frequency of outbound connections
- Gradual increase in both `sbytes` and `spkts`
- Abnormal flow patterns to new destinations

Worm attacks function autonomously to distribute themselves quickly through networks by taking advantage of system weaknesses. The detection of worms depends on temporal flow analysis because these threats create rapid bursts of connections to multiple targets during short periods. The identification of this behavior depends heavily on two key features: `ct_dst_src_ltm` which tracks connections between the same source and destination within a time window and overall connection rate metrics. A high frequency of similar flows from a single host indicates worm-like propagation.

The analysis of low-level traffic behaviors through flow-based anomaly detection enables the identification of various cyberattacks. The extraction and monitoring of statistical features including packet counts and flow duration and byte volumes and TTL values and connection states enables the detection of known and unknown attacks including DoS and exploitation attempts and backdoor communications and reconnaissance efforts and worms. The distinct patterns in flow data allow models to generalize effectively in real-world environments because each attack type produces unique patterns. The specific flow metrics directly match the behavioral signatures of different attack categories which make flow-based anomaly detection suitable for modern network defense systems.

CHAPTER 5: METHODOLOGY AND ARCHITECTURE

The design of a robust Intrusion Detection System (IDS) using artificial intelligence requires a carefully orchestrated methodology that spans data preparation, model selection, architectural decisions and system integration. The entire methodological framework used for this thesis is presented in this chapter which details both conceptual and technical aspects of the hybrid detection system.

The main goal of the proposed methodology is to create an IDS system that offers scalability alongside accurate real-time anomaly detection and threat classification capabilities. The system implements unsupervised and supervised learning techniques through two stages: an autoencoder-based anomaly detection module and a LightGBM-based attack classifier. These system components were designed to operate on flow-based network data for identifying harmful network traffic and when possible, determining the attack type.

The methodology implements a structured data preprocessing pipeline which starts with categorical encoding, numerical normalization and classification label handling. The entire system maintains uniform preprocessing steps for both autoencoder and LightGBM classifier operations to maintain data compatibility and ensure pipeline modularity. The chapter demonstrates how label encoding and one-hot encoding are implemented according to model needs and target applications such as regression versus classification.

The architectural components of the system receive detailed explanation after this section. The autoencoder module is trained only on benign traffic data to discover normal behavior patterns which it uses to detect outliers through reconstruction errors. The LightGBM classifier operates differently from the autoencoder because it learns from attack data labels to detect specific attack categories while delivering interpretable attack details. The Security Bot pipeline serves as the third architectural element which combines both components to perform automated prediction followed by response generation and logging functions.

This chapter explains design choices together with hyperparameter selections and details how the dynamic threshold system adapts to separate normal from anomalous system behavior. A section on model generalization together with computational efficiency and real-time feasibility evaluation is included to validate the methodology under operational limitations.

The lightweight implementation choices together with optimized inference strategies support virtualized environments deployment of hybrid architecture. The methodology concludes with evaluation and interpretation procedures for models which create a

foundation for the following chapter about implementation details and performance assessment.

5.1 SYSTEM OVERVIEW

The system was built to serve as a hybrid intrusion detection system which applies unsupervised and supervised machine learning approaches to network threat detection in real time. The system has been designed to solve IDS system problems including the inability to detect unknown threats and it provides a flexible solution that can be implemented in virtualized systems including isolated testbeds or production VMs.

As shown in **Figure 5.1** the system has four main stages that are used for data acquisition, preprocessing, anomaly detection through autoencoders and supervised LightGBM attack classification and IP blocking. These components are integrated into a unified processing pipeline, referred to as **SecurityBot**, which handles the end-to-end flow of live network traffic through the detection system.

Packet level data is captured from a live network interface using Scapy [18]. Each packet is combined into a flow-based representation by using a 5-tuple identifier (source IP, destination IP, source port, destination port, and protocol). The flow abstraction method reduces the computational complexity of the system but maintains the required level of detail to study behavioral patterns over time. Each flow has a set of features such as `sbytes`, `dbytes`, `spkts`, `dpkts`, `sttl`, and `dur` that are selected and used as inputs for downstream models.

The SecurityBot uses a trained autoencoder model to detect normal network flow behavior during unsupervised learning. The model attempts to reconstruct input vectors, and the reconstruction error is measured for each flow. Any flow that exceeds this threshold is marked as an anomaly. This phase allows the system to detect unknown attacks or zero-day behaviors, where no labeled data exists.

After a flow is marked as anomalous it is passed to a supervised classification module powered by a pre-trained LightGBM model. The module is responsible for classifying the anomaly into one of the predefined attack types (e.g., DoS, Fuzzers, Backdoor). The LightGBM model operates on the same set of features and benefits from label encoding and feature scaling consistent with the autoencoder. The classifier thus provides semantic interpretation of anomalies and helps prioritize responses.

In the last stage of the system, suspicious flows are logged, classified and automatically blocked by iptables rules. Anomalies and their classifications are logged in a structured format for analysis and optional alerts can be configured to trigger administrative response.

The high-level system architecture shows the data and control flow between components, as well as the detection-response bidirectional interaction.

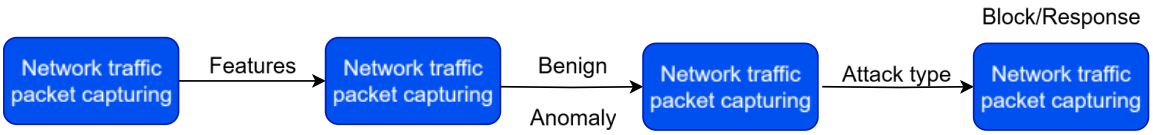

Figure 5.1: High Level Architecture Diagram.

5.2 MODEL SELECTION AND TRAINING

Model development required extensive testing of various machine learning architectures including supervised and unsupervised models. The final hybrid system consists of two main components which include an autoencoder-based anomaly detector and a LightGBM classifier. This section explains the selection of models, training approach and the evaluation process that resulted in the final architecture.

5.2.1 Explored Models

During the early stages of experimentation, a number of models were evaluated to determine their suitability for network anomaly detection and classification:

Unsupervised Models

- **Autoencoder**

  The simple autoencoder is an unsupervised model designed to learn the normal patterns in network traffic data by reconstructing the input flow features. This architecture is relatively simple, comprising an encoder and a decoder, both composed of fully connected layers with ReLU activations in the hidden layers. The autoencoder is trained only on normal traffic to learn the statistical patterns of benign behavior, and reconstruction errors on test data are used to flag anomalies.

Training Configuration for the Autoencoder

The model was trained with the following configuration:

- **Encoder-Decoder Architecture**: The encoder transforms input data into a reduced-dimensional space through two dense layers which are followed by Dropout layers to reduce overfitting. The first dense layer containing 64 neurons with sigmoid activation identifies high-level features before the second dense layer with 32 neurons refines these features. The final dense layer known as the latent space contains latent_dim neurons which store the compressed

version of the input data. The compressed encoding from the encoder feeds into the decoder which works to reconstruct the original data by reducing reconstruction errors.

- **Activation Functions**: Sigmoid for hidden and activation layers for the output layer to ensure that the reconstruction error is minimized.
- **Loss Function**: Mean Squared Error (MSE), which calculates the squared difference between the input and the reconstructed output.
- **Optimizer**: Adam optimizer [3] with a learning rate of 0.00005.
- **Epochs**: Training was performed for 25-50 epochs, with early stopping [4] based on validation loss to prevent overfitting.
- **Batch Size**: 256.

The threshold for anomaly detection was set dynamically by analyzing the reconstruction errors in the validation dataset. Anomalous flows were flagged when their reconstruction error exceeded a predefined threshold, which was adjusted based on the specific detection requirements and the desired balance between false positives and false negatives.

- **AnomalyDAE:**

  The dual autoencoder system AnomalyDAE uses two separate encoders to model both structural data and node attributes independently. The model received testing during initial experimentation, but researchers eliminated it because it required more computational resources and added complexity without delivering better results than the basic autoencoder. The dual-layer structure of this model proved beneficial for graph-based anomaly detection, yet it failed to deliver optimal results for flow-based intrusion detection on the chosen datasets.

Key Differences from Simple Autoencoder

- **Dual-Layer Structure**: AnomalyDAE splits the model into two encoders to separately model structural data (network flow structure) and node attributes (IP addresses, ports, etc.).
- **Increased Complexity**: The additional layer made the model computationally intensive and more prone to overfitting, without offering significant improvements over the simpler autoencoder.

Supervised Models

- **LightGBM:**

  The LightGBM classifier served as the attack classification stage tool because it supports large datasets and multiclass classification and delivers fast training and inference speeds. The LightGBM framework uses decision trees to build an ensemble that improves predictions through iterative processes. The framework excels at processing imbalanced datasets because intrusion detection systems usually have more normal traffic than attack instances.

Training Configuration for LightGBM

- **Objective**: Binary classification for the presence or absence of attacks, with a multiclass extension for specific attack types.
- **Boosting Type**: GBDT (Gradient Boosting Decision Trees).
- **Hyperparameters**:
  - **Learning Rate**: 0.05
  - **Number of Leaves**: 31
  - **Bagging Fraction**: 0.8
  - **Feature Fraction**: 0.8
- **Hyperparameter Tuning**: Performed using **GridSearchCV** to optimize key parameters such as num_leaves, learning_rate, and n_estimators.

LightGBM was trained using both binary labels (normal vs. attack) and multiclass labels (10 distinct attack types) from the UNSW-NB15 dataset. The model performed well in classifying attacks and achieved high accuracy across several attack categories. It also demonstrated excellent scalability, which is crucial for real-time deployment scenarios.

Model Selection Strategy

The simple autoencoder was chosen for unsupervised anomaly detection due to its simplicity and effectiveness in learning the normal traffic distribution. Although AnomalyDAE showed potential, its complexity outweighed the performance benefits for the task at hand, making the simpler autoencoder a more efficient choice. For attack classification, LightGBM provided a very good balance of accuracy, training time, and interpretability.

5.3 DATA PREPROCESSING PIPELINE

Machine learning workflows require preprocessing as an essential step because real-time network anomaly detection systems need to transform raw network traffic data before machine learning models can process it effectively. The preprocessing pipeline of this thesis was created to provide both unsupervised (autoencoder) and supervised (LightGBM) models with standardized high-quality data inputs. The following section describes the preprocessing operations which were applied to the dataset including feature selection, encoding, scaling, class balancing and missing value handling.

5.3.1 Data Cleaning and Handling Missing Data

The preprocessing pipeline starts with data cleaning to eliminate all inconsistencies and missing values. Real-world network traffic data contains missing or corrupted entries because it captures live network data. UNSW-NB15 and Kyoto 2006+ datasets handled missing values through the following approach:

- **Missing Data Imputation**: The preprocessing stage used imputation methods to replace missing feature values. The imputation of numerical missing values used either the mean or median value from the respective column based on the feature distribution. The most frequent category in each column served as the imputation value for missing categorical data points.
- **Removal of Incomplete Rows**: The models received only complete rows because any row with more than 20% missing data was discarded to prevent model contamination. The decision to discard rows followed the requirement for using reliable and complete data in training and testing processes.

5.3.2 Categorial Feature Encoding

UNSW-NB15 and Kyoto 2006+ datasets contain categorical variables which include proto (protocol), state (connection state) and service. Tree-based models such as LightGBM cannot directly handle categorical data. Two encoding techniques were used to address this issue:

- **Label Encoding**: Assigns numerical labels to each unique category found in a feature. The method was used for features that have an ordinal relationship such as state (e.g., SYN, FIN, ESTABLISHED). Label encoding is a compact form of conversion, as each category is represented by a single integer. The autoencoder model required compact representations to minimize input dimensionality thus the encoding technique was used.
- **One-Hot Encoding**: It was used for features that do not have an inherent ordinal relationship such as proto (protocol type) or service. This method creates binary

columns for each unique value in the categorical feature, effectively expanding the feature space. This encoding was used for the LightGBM classifier, as decision tree-based models perform better with non-ordinal features treated as independent variables.

5.3.3 Feature Scaling and Normalization

Machine learning models that use distance metrics or gradient-based optimization need feature scaling because it is essential for autoencoders and neural networks. The UNSW-NB15 dataset contains features with extreme magnitude differences because `spkts` vary between several thousand and a few units, while dur spans from seconds to minutes. Models tend to favor features with larger magnitudes when unscaled, which results in poor convergence and suboptimal performance.

The numerical features required Min-Max normalization as a scaling method. The numerical features `sbytes, dbytes, spkts, dur` and others received Min-Max scaling which transformed their values into the range [0, 1]. range of feature values. Min-Max scaler transforms features into a specific range, typically between **0 and 1**. The purpose of scaling is to ensure that each feature contributes equally to the model, preventing features with larger ranges from dominating the learning process. Neural network autoencoders require this scaling because it prevents any feature from controlling the learning process of the model. The transformation method accelerates training because it reduces the variance of gradients during backpropagation.

The feature scale does not affect LightGBM because tree-based models remain unaffected by feature scale. StandardScaler [19] was used to scale all features because it provides a standardization process that sets both mean and standard deviation to 1 for model comparison purposes.

5.3.4 Data Splitting

To evaluate the model's performance, the dataset was split into training, cross-validation and testing sets.

- **Training set**: 70% of the datasets were used to train the models, containing both benign and malicious traffic. The malicious traffic data from the datasets was used to train the LightGBM classifier. The autoencoder was trained only on normal traffic data, which were the majority of both datasets.
- **Cross-validation set**: 20% of the data was used to tune the models' hyperparameters and monitor performance during training.

- **Testing set**: The final 10% was used to test the models' ability to generalize on previously unseen data, to prevent overfitting.

5.3.5 Class Imbalance

Network intrusion detection systems face class imbalance problems because attack traffic occurs much less frequently than normal traffic. The minimal ratio of malicious flows to benign flows in datasets like UNSW-NB15 and Kyoto 2006+ creates biased model predictions. The handling of class imbalance leads models to predict the majority class (normal traffic) exclusively, while failing to detect rare attack instances.

Stratified Sampling

Stratified sampling is a sampling technique used in machine learning to ensure that each subgroup of a population is fairly represented in the sample. The implementation of stratified sampling during data splitting helped reduce class imbalance. This approach maintains the normal to attack class ratio throughout the training, validation and testing datasets. The model gains knowledge about both common network traffic and attack behaviors through stratified sampling which maintains equal attack and normal traffic representations in each subset. Especially in classifier's training require this step because it ensures both benign and malicious instances remain available for model training and validation.

SMOTE (Synthetic Minority Over-sampling Technique)

The implementation of stratified sampling does not eliminate class imbalance because specific attack categories including shellcode, backdoors and worms remain underrepresented in the training data. The solution to this problem was addressed through the implementation of SMOTE [20] which generates synthetic instances to increase minority class representation.

SMOTE works by:

- Selecting a minority class sample.
- The system identifies the nearest neighbors which belong to the same class.
- The system produces new samples through the process of interpolating between the selected instance and its neighboring instances.

The system generates new attack flow samples through the creation of synthetic data points which are derived from the feature space of minority class data. The purpose of this process is to enhance minority class diversity which prevents overfitting and enables better decision

boundary learning. SMOTE was used to balance attack traffic during preprocessing before feeding the data to the LightGBM classifier. Through this approach the classifier gained knowledge about a more balanced distribution of attack types without developing excessive dependence on normal traffic data. The introduction of synthetic examples through SMOTE improved LightGBM's ability to detect minority attack classes while maintaining its accuracy in the majority class. SMOTE's oversampling is illustrated in **Figure 5.2**.

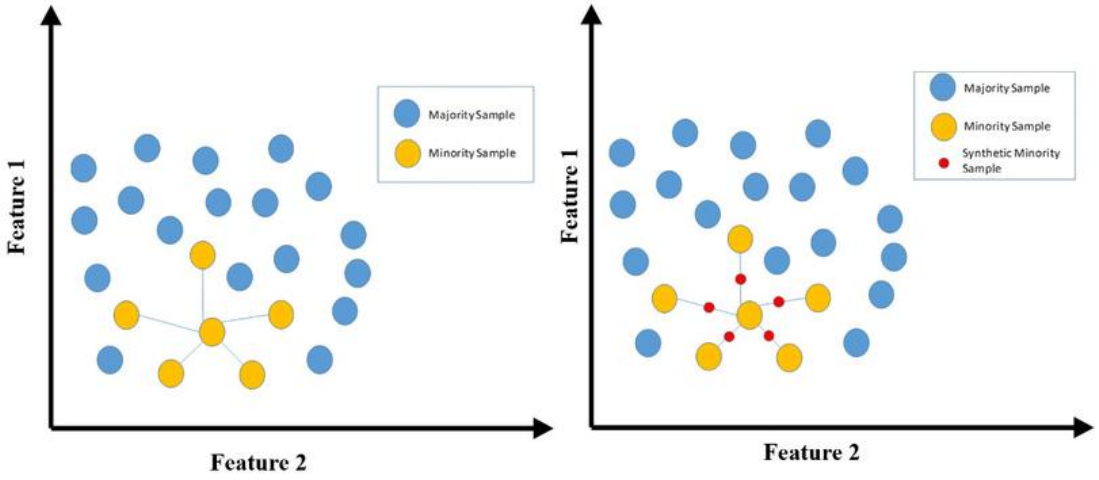

Figure 5.2: Illustration of SMOTE oversampling. [21]

5.4 INTRUSION DETECTION SYSTEM

In this section, we will discuss the integration and deployment process of the Hybrid Intrusion Detection System (IDS) that combines Autoencoder-based anomaly detection and LightGBM-based attack classification. This process includes how the system was designed to handle real-time network traffic, continuously monitor and detect anomalies, and classify network behaviors, all while ensuring efficient performance and scalability. The architecture of the intrusion detection system (IDS) is based on a hybrid approach that combines unsupervised anomaly detection using the Autoencoder and supervised attack classification using LightGBM. The system works by capturing network traffic, detecting anomalies, and classifying attacks. **Figure 5.3** shows the high-level architecture of the intrusion detection system.

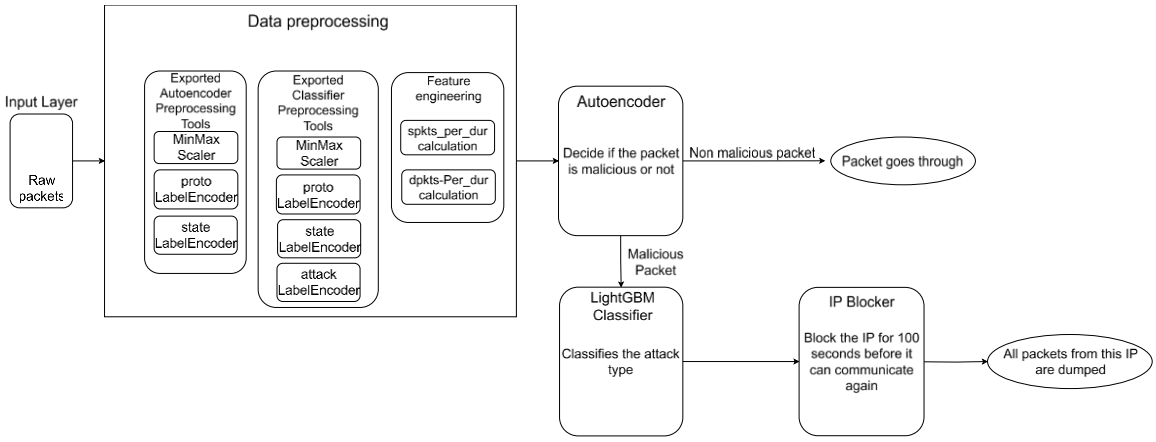

Figure 5.3: Intrusion Detection System Architecture.

Real-time Traffic Capture: Network traffic is captured in real-time using packet sniffing tools. The data is then processed to extract relevant features for anomaly detection and attack classification.

Feature Extraction: From the captured packets, key features are extracted that are relevant for the anomaly detection and classification tasks. These features are critical for the subsequent processing steps, and ensure the model has the necessary data to detect deviations from normal behavior.

Preprocessing: Once the features are extracted, they undergo preprocessing to prepare them for input into both the Autoencoder and the LightGBM classifier: Features are normalized using a MinMaxScaler to ensure they are within the same range, making them suitable for both models. Categorical variables such as proto and state are encoded using LabelEncoder, transforming them into numerical values that the models can process.

Anomaly Detection (Autoencoder): The Autoencoder detects anomalies by learning normal traffic patterns during training. If the reconstruction error between input and output exceeds a dynamic threshold, the flow is flagged as an anomaly.

Attack Classification (LightGBM Classifier): Once an anomaly is detected, the LightGBM classifier classifies the anomaly into one of several attack types (e.g., DoS, Exploits, DDoS). It is trained using labeled data and provides a prediction for each anomaly.

Response Mechanism (IP Blocking): If an attack is confirmed, the system triggers an automated response, blocking the malicious IP using a firewall (e.g., iptables) to prevent further damage.

5.5 AUTOENCODER ARCHITECTURE

The autoencoder model functions as the unsupervised anomaly detection module within the hybrid intrusion detection system. This section explains the autoencoder architecture by describing its layers, activation functions, optimization methods and training approach. The autoencoder model detects abnormal network traffic patterns which could represent unknown attacks or system anomalies. **Figure 5.4** shows the architecture diagram of the autoencoder.

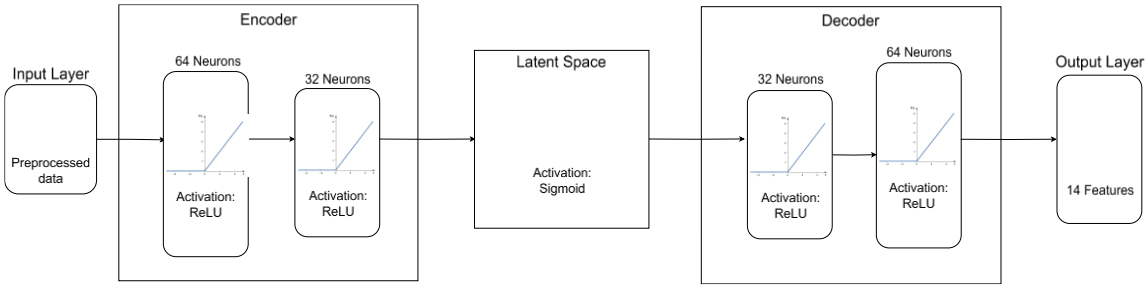

Figure 5.4: Autoencoder architecture diagram.

5.5.1 Autoencoder Architecture Design

The autoencoder model in this thesis consists of an encoder-decoder structure, which is designed to learn a compressed, lower-dimensional representation of network flow data. The architecture can be divided into two main parts:

Encoder:

- The encoder maps the input data into a latent space, capturing the most significant features of normal traffic while discarding less important information.
- It consists of two dense layers with ReLU activation functions (see APPENDIX D: UNDERSTANDING ACTIVATION FUNCTIONS IN MACHINE LEARNING) and a latent layer at the bottleneck. This representation is the reduced feature set that the model uses to perform anomaly detection. The latent space has a smaller number of neurons to force the model to learn the most important features from the data.

Latent Space:

- The latent space represents the compressed, low-dimensional representation of the input data. This is the “bottleneck” of the model, where the most significant features of the input data are retained in a reduces format.

- The Sigmoid function is applied to the latent space to ensure that the values lie between 0 and 1.

Decoder:

- The decoder attempts to reconstruct the input data from the compressed latent representation. It uses the same architecture as the encoder but in reverse, expanding the features back to the original input size.
- The reconstruction error (e.g., Mean Squared Error) between the input and the reconstructed output is used to flag anomalies.

5.5.2 Advantages of Autoencoder

Autoencoder was selected, bearing in mind it has the following advantages:

- **Detection of Unknown Attacks**: Unlike signature-based detection methods, which rely on predefined patterns of known attacks, autoencoders can detect zero-day attacks and novel threats. Since the model is trained to understand only normal traffic patterns, it can identify any deviation from these patterns as an anomaly, even if the attack has never been encountered before.
- **Robust to Noise**: Autoencoders can be robust to noisy data, making them suitable for real-world applications where network traffic may include various inconsistencies or errors. The model's ability to focus on the key features of the input data helps to reduce the impact of noise and ensures that the reconstruction is accurate even with imperfect data.
- **Scalability and efficiency**: The real-time applications benefit from using Autoencoders because shallow models operate with high computational efficiency. The system can process datasets containing millions of records which makes it appropriate for real-time anomaly detection in high-traffic network environments. The autoencoder delivers fast reconstruction results which enables it to provide immediate feedback during live system deployment.

5.5.3 Training and Optimization

The autoencoder received normal traffic data for training to achieve its main objective of minimizing the reconstruction error between input data and its latent space representation. The training process is outlined as follows:

- **Loss Function**: The Mean Squared Error (MSE) function measured the difference between original inputs and reconstructed outputs during training. The MSE loss

function was chosen because it detects major deviations in expected traffic behavior effectively.

- **Optimizer**: The Adam optimizer served as the training optimizer with a learning rate set at 0.001. The Adam optimizer optimizes loss function minimization while ensuring training process stability.
- **Epochs and Early Stopping**: The model ran for 100 maximum epochs while using validation loss for early stopping to avoid overfitting.

The model determined its anomaly detection threshold through validation set reconstruction error calculations. Through this approach the model learned to modify its anomaly detection sensitivity according to current network traffic patterns.

5.5.4 Anomaly Detection

Once trained, the autoencoder model was able to detect anomalies by comparing the reconstruction error between the input and the output. A high reconstruction error indicates that the model was unable to accurately reconstruct the input, suggesting that the flow is an anomaly.

- **Anomaly Threshold**: The threshold for what constitutes an anomaly was set dynamically, depending on the mean and standard deviation of the reconstruction errors observed during training. A flow is flagged as anomalous if its reconstruction error exceeds a predefined threshold.
- **Dynamic Thresholding**: Unlike a static threshold, dynamic thresholding adapts based on the current data distribution, ensuring that the model can detect both subtle and drastic anomalies based on the ongoing network behavior.

The dynamic thresholding approach plays a crucial role in upscaling the anomaly detection capabilities of the autoencoder model. Unlike a static threshold, which is fixed and does not adapt to changes in network traffic, dynamic thresholding adjusts based on the statistical properties of the reconstruction errors observed during training. This method computes the threshold based on the mean and standard deviation of the reconstruction errors from the training data. By setting the threshold dynamically, the system ensures that the anomaly detection process remains flexible and sensitive to fluctuations in network behavior. For example, during periods of low activity, a lower threshold may be set to ensure subtle anomalies are detected, while during high-traffic periods, the threshold may increase to prevent flagging normal fluctuations as anomalies. This adaptive mechanism improves the model's ability to detect both minor and major anomalies without being oversensitive or missing critical deviations. Consequently, dynamic thresholding ensures that the autoencoder can effectively handle varying network conditions and accurately identify unknown threats.

5.6 LIGHTGBM MULTICLASS CLASSIFIER

The LightGBM classifier functioned as the fundamental model for attack classification throughout this research. The LightGBM gradient boosting framework, developed by Microsoft, demonstrates exceptional performance when processing extensive datasets with numerous features. The algorithm demonstrates superior performance when dealing with imbalanced data because network traffic datasets typically contain much more normal traffic than attack traffic. This section describes the training setup and hyperparameter choices and how LightGBM integrates into the complete system framework. **Figure 5.5** depicts the LightGBM architecture.

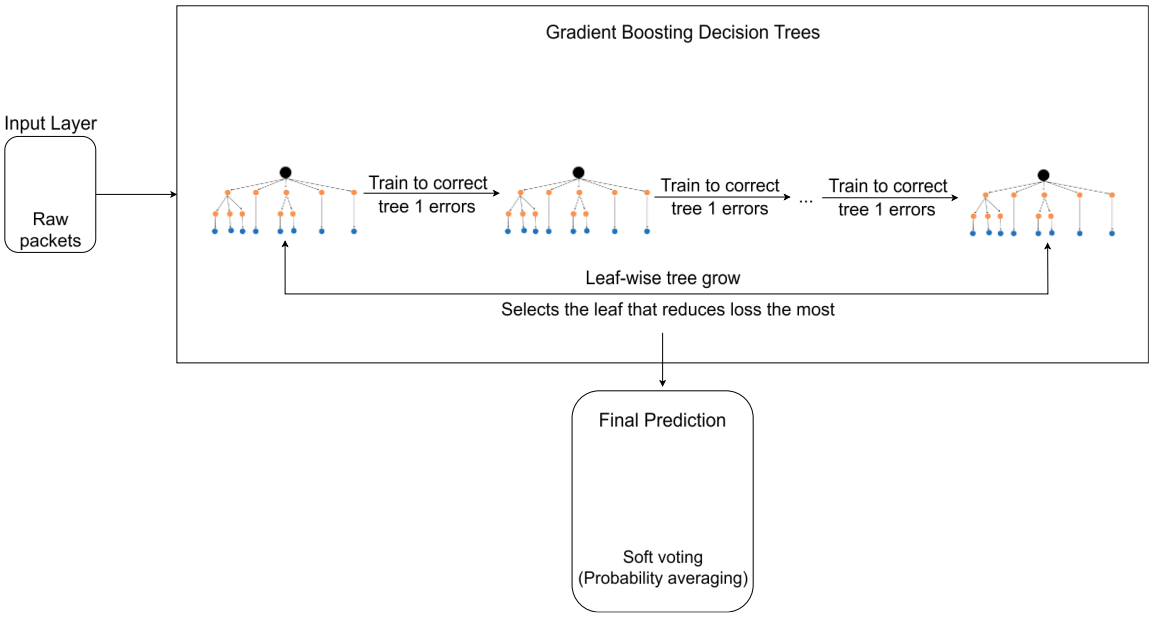

Figure 5.5: LightGBM Architecture.

5.6.1 LightGBM Classifier Architecture

The LightGBM classifier operates through training multiple decision trees. Each layer tries to compensate for the errors made by preceding trees. The training process relies on boosting which enables each new tree to minimize the residual error between predicted and actual values. The training process stops when either a specified number of boosting rounds is reached or when the validation error no longer improves.

The architecture of the LightGBM model can be described as follows:

Input Features: The input features for LightGBM come from the preprocessed data, including the selected network flow features (e.g., `sbytes, dbytes, spkts, dur,` etc.). These features are used to learn the decision boundaries between normal and attack

traffic. One-hot encoding and label encoding were used to transform categorical features (such as proto and state) into numerical representations.

Gradient Boosting Trees (GBDT): LightGBM constructs a sequence of decision trees, where each tree is trained to minimize the residual error from the previous tree. Unlike traditional decision trees, which are built independently, GBDT builds trees sequentially to focus on the errors that earlier trees made. LightGBM also uses a leaf-wise growth approach, where the model selects the tree leaf that minimizes the loss function most, allowing it to focus on the hardest examples. This approach can lead to better accuracy but may result in overfitting if not properly regularized.

Loss Function: LightGBM uses SoftMax loss, for multiclass classification, to handle multiple classes, such as DoS, DDoS, Exploits, etc. The SoftMax function converts the outputs of each tree into a probability distribution across all classes. The objective function was set to multiclass log loss during training, which helps the model to learn how to distinguish between multiple attack types effectively.

Regularization: L1 and L2 regularization are applied to the model to prevent overfitting, which is particularly important in high-dimensional datasets like network traffic, where many features are used. The regularization terms penalize overly complex trees and ensure that the model generalizes well to unseen data

Hyperparameter Tuning: Hyperparameters such as learning rate, number of leaves, tree depth, and boosting rounds were optimized during training using GridSearchCV or RandomizedSearchCV. Proper tuning helps to balance model complexity and performance, ensuring that the model is not too complex (which could lead to overfitting) or too simple (which could result in underfitting).

Ensemble of Trees: The final prediction is made by aggregating the predictions of all decision trees in the model. The aggregation process may involve soft voting (averaging probabilities) or hard voting (selecting the majority class). In this case, soft voting is used, where the model's output is based on the average probability across all trees.

Class Weighting: Class weighting is an essential feature of LightGBM for handling imbalanced datasets (normal vs. attack traffic). By assigning a higher weight to the minority class (attack traffic), LightGBM compensates for the disproportionate representation of normal traffic, helping the model to focus more on detecting rare attack types.

5.6.2 Advantages

LightGBM was selected because of:

- **Scalability**: It can efficiently handle large datasets with millions of instances, which are common in network traffic data.
- **Performance**: LightGBM outperforms many machine learning algorithms in terms of speed and accuracy, especially on imbalanced data, making it ideal for intrusion detection systems where the normal traffic far outweighs attack traffic.
- **Interpretability**: LightGBM provides valuable insights into feature importance, allowing us to interpret which network flow features are most influential in detecting attacks.

Additionally, LightGBM's ability to handle class imbalance through class weighting and subsampling makes it well-suited for the attack classification task, where malicious traffic constitutes a small portion of the overall dataset.

5.6.3 Hyperparameter Selection and GridSearchCV

To achieve optimal performance, GridSearchCV was used for hyperparameter tuning, systematically testing different combinations of the following hyperparameters:

- **Learning Rate**: 0.05, controlling how much to adjust weights at each step of boosting.
- **Number of Leaves**: 31, determining the complexity of individual trees.
- **Boosting Rounds**: 500, specifying the number of boosting iterations.
- **Bagging Fraction and Feature Fraction**: Both set to 0.8, improving model regularization and robustness.
- **Max Depth**: 15, ensuring each tree does not become too deep, preventing overfitting.

After trial and error, this exhaustive search found the best-performing combination of hyperparameters based on cross-validation accuracy.

5.6.4 Attack Classification

The LightGBM classifier takes control after the autoencoder detects network traffic anomalies to identify their attack types. The classification process relies on supervised learning because the model received training from labeled data that includes attack categories. The multiclass classification system assigns distinct labels to each attack type. The LightGBM model predicts the class label for anomalous traffic by applying learned features from its training process to identify the attack type. The two-step process of

anomaly detection through autoencoders followed by LightGBM classification enables both the discovery of new threats and the precise identification of recognized attack types.

CHAPTER 6: SYSTEM PROTOTYPE

The hybrid IDS system described in this thesis combines Autoencoder-based unsupervised anomaly detection with LightGBM-based supervised attack classification. This chapter describes the technical implementation of the system design and all development and deployment steps for the IDS. The chapter describes the entire process starting from a virtualized environment setup for testing through data collection, preprocessing and model training and Autoencoder and LightGBM model integration. The system functions in real-time by collecting network traffic while identifying abnormal traffic patterns which it then classifies as particular attack types. The system used network traffic features from real-time data along with advanced machine learning methods to detect known and unknown attacks. The implementation process encountered performance challenges and difficulties which are discussed in this chapter including class imbalance problems and real-time processing requirements and model evaluation methods.

In this chapter, we will cover the following:

1. **System Environment**: A description of the development environment, including the operating system, network interface, Python version, and other dependencies.
2. **Tools and Libraries**: An explanation of each tool and library used, including why they were chosen and how they interact in the system.
3. **Virtual Machine Setup**: Detailed setup for the virtual environment used for testing the IDS.
4. **Data Collection and Attack Simulation**: Details of the UNSW-NB15 dataset, virtualized test setup, and how network attacks were simulated. Also in this section, some necessary network background will be quickly reviewed.
5. **Data Preprocessing and Feature Engineering**: An explanation of the preprocessing steps, including label encoding, feature scaling, and feature extraction from network traffic.
6. **Autoencoder Model Training**: A detailed explanation of the model training process for the anomaly detection system.
7. **LightGBM Model Training**: A detailed explanation of the model training process for the attack classification system.
8. **Intrusion Detection System**: How the two models were integrated to work in tandem for real-time detection and automated attack classification.
9. **Challenges and Limitations**: The difficulties encountered during implementation and strategies used to address them.

10. **Future Work**: Suggestions for improvements and future developments of the IDS.

This chapter serves as a comprehensive guide to the implementation details of the hybrid IDS, providing insight into how the system was built, trained, and deployed for real-world intrusion detection.

6.1 SYSTEM ENVIRONMENT

The Hybrid Intrusion Detection System (IDS) received a carefully designed system environment to support its training, testing and deployment processes. The IDS operated in a virtualized environment through two main virtual machines (VMs) which ran Ubuntu and Kali Linux. The system consisted of two virtual machines where Ubuntu functioned as the victim machine and Kali Linux operated as the attacker machine to conduct network attacks. The following section explains the complete system environment setup. **Figure 6.1** shows the final environment used for testing this thesis.

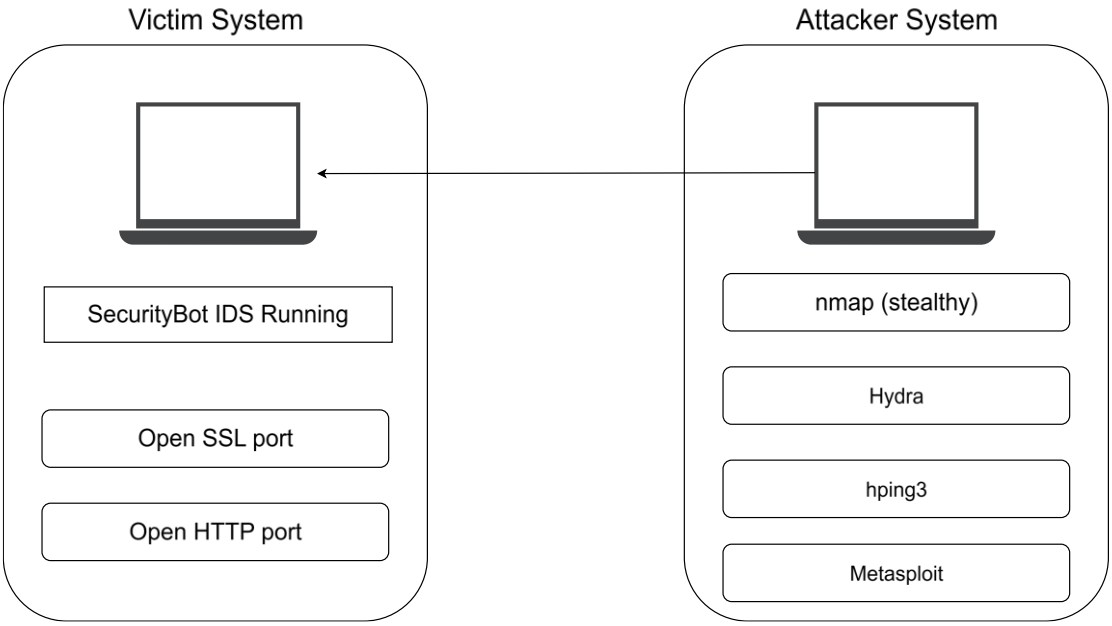

Figure 6.1: System environment overview.

6.1.1 Operating System Setup

The two primary operating systems used for testing the intrusion detection system were:

1. **Ubuntu (Victim Machine)**:
   - **Ubuntu Operating System**: Ubuntu 20.04 LTS was used for the victim machine. This operating system was chosen for its stability, broad support

in networking environments, and ability to easily configure network interfaces and services for simulation purposes.

- **Network Traffic Simulation**: Ubuntu served as the environment for generating normal network traffic. Common network protocols such as HTTP, FTP, and SSH were used to simulate real-world traffic patterns.
- **Hosted Services**: The Ubuntu machine hosted several services (such as Web Servers, SSH, etc.), generating network traffic that was monitored and analyzed by the IDS.

2. **Kali Linux (Attacker Machine)** [22]:
   - **Kali Operating System**: Kali Linux 2021.4 was used as the attacker machine. Kali Linux is a popular operating system for penetration testing, containing a comprehensive suite of security tools such as Metasploit, Nmap, and Hydra, which were utilized to simulate various types of attacks on the victim machine.
   - **Attack Simulation**: Kali Linux was responsible for generating attack traffic, simulating Denial-of-Service (DoS), DDoS, Reconnaissance, and Exploits using the tools. These attacks were targeted at the victim machine (Ubuntu) to test the IDS.

6.1.2 Network Interface Configuration

The network interface configuration was essential to the setup for monitoring and analyzing traffic. The configuration included:

Network Interfaces: The victim machine (Ubuntu) used the ens33 network interface for communicating with the attacker machine (Kali Linux). The network traffic between the two VMs was monitored in real-time by the intrusion detection system, which continuously sniffed the network traffic.

Virtual Network Setup: The VMs operated within the same virtual network to create a realistic environment. The two VMs communicated through a private network established within the VMware or VirtualBox virtualization environment for smooth data exchange. The virtual network ensured that the Ubuntu victim machine was isolated from external traffic, meaning the IDS could focus solely on internal traffic between the victim and attacker machines

Traffic Monitoring: The system used the integrated network interface card (NIC) on the host machine to monitor the flow-based network traffic generated by the VMs. The interface allowed the IDS to capture, process and analyze data between the victim and attacker systems.

6.1.3 Python Version

The entire system was developed in Python 3.8 due to its compatibility with the required libraries for machine learning, data processing, and network traffic analysis. Python 3.8 was chosen as it supports a wide range of libraries and has stable compatibility with the latest versions of TensorFlow [23], LightGBM, and Scikit-learn [19]. The environment was set up in a virtual environment (venv) to avoid conflicts between system-wide packages and project-specific dependencies.

6.2 TOOLS AND LIBRARIES

The Hybrid Intrusion Detection System (IDS) developed in this thesis depends on Python libraries and tools for network traffic capture, feature extraction, machine learning, and model management. The system requires these tools to operate effectively for real-time anomaly detection and attack classification. The following section explains the tools and libraries used in the system's development in detail.

6.2.1 Scapy – Packet Sniffing and Flow Construction

Scapy is an essential tool used in the system for real-time packet sniffing and flow construction. Scapy allows for packet-level analysis and flow-based feature extraction, making it an ideal tool for network intrusion detection.

- **Real-time packet capture**: Scapy captures packets from the network interface in real-time. It decodes these packets and allows for detailed inspection of each one, including IP addresses, protocol types, packet size, and more. The system captures relevant network flows (identified by the 5-tuple) between source and destination IPs, source and destination ports, and protocol.
- **Flow construction**: The captured packets get grouped into flows through Scapy using the 5-tuple as the aggregation criteria. The flow-based intrusion detection process requires this step because it combines packet-level data into flows for simpler analysis. The system generates flow-based features including packet count and byte volume and TTL and more which serve as inputs for both anomaly detection and attack classification.
- **Advantages**: Scapy allows custom packet analysis, making it flexible for extracting the necessary features for the IDS. It supports real-time sniffing, making it suitable for intrusion detection in live environments.

Alternatives like Wireshark or tcpdump are also capable of sniffing packets but were not used because they are more geared towards manual inspection and are not easily automated for real-time analysis. Scapy provides more flexibility for programmatic packet processing and flow extraction, which is crucial for the real-time nature of this IDS.

6.2.2 TensorFlow/Keras for Autoencoder anomaly detection

TensorFlow (with Keras as its high-level API) is the framework used to implement the Autoencoder for anomaly detection. Autoencoders are a type of neural network used to learn efficient representations of data, and they are particularly effective at unsupervised anomaly detection.

TensorFlow

The machine learning framework TensorFlow enables users to develop models through its complete workflow starting from training until deployment. The system enables efficient processing of big data sets and supports multiple machine learning algorithms and scales across GPU and distributed computing systems. It operates on the concept of tensors (multi-dimensional arrays), which are used to represent data. TensorFlow processes these tensors to perform various mathematical operations required for model training and inference.

Key Features

- **GPU Acceleration**: TensorFlow is optimized for running on GPUs enabling speed improvements in model training. This is crucial when dealing with large datasets or training models in real-time where it is needed to process as fast as possible.
- **Distributed Training**: TensorFlow supports distributed training across multiple machines, which is useful when training on very large datasets or cloud-based development.

It was used in this thesis to train the Autoencoder model for anomaly detection. The framework's ability to efficiently process large amounts of data and accelerate model training using GPU resources played a big role in ensuring real-time performance of the IDS.

Keras

Keras serves as an open-source high level neural network API which allows users to quickly experiment with deep learning models. The Keras API simplifies TensorFlow operations through its user-friendly interface which enables model development and training and evaluation processes. While it started as an independent library, it has now been integrated into TensorFlow as its official high-level API. This integration makes TensorFlow more user-friendly, particularly for those who are new to deep learning.

Key features

- **Simplified Model Building**: The Keras framework enables users to construct deep learning models through its straightforward interface which supports

Dense, Convolutional and Recurrent layers. A neural network can be built through simple code by stacking multiple layers.

- **Pre-built components**: Keras provides numerous pre-built components which include activation functions, optimizers and loss functions and layers to help users test different architectures and hyperparameters.
- **Model Training and Evaluation**: The training and evaluation process becomes straightforward through Keras because users can easily specify epoch numbers, batch sizes and validation data. The training process includes early stopping callback functionality which helps prevent overfitting from occurring.

In this thesis, Keras was used to implement and train the Autoencoder model. The Keras interface made it easy to develop the system quickly and TensorFlow integration enabled GPU acceleration for efficient model training.

6.2.3 Scikit learn for Data Preprocessing and Evaluation

Scikit-learn is a versatile library used for data preprocessing, model evaluation, and performance metrics. It is widely used for its flexibility and ease of use, especially when dealing with machine learning tasks such as classification and regression.

Data Preprocessing

- **Label Encoding**: The LabelEncoder from Scikit-learn converted categorical variables like protocol and state into numerical values which both Autoencoder and LightGBM could utilize.
- **Feature Scaling**: MinMaxScaler from Scikit-learn was used to normalize numerical features like packet counts, flow duration, and byte volume. This ensures that the features are on the same scale, improving the model's performance.

Model Evaluation

- Accuracy, precision, recall, and F1-score were used to evaluate both the Autoencoder and LightGBM models.
- Cross-validation was used to assess model performance and avoid overfitting, ensuring that the models generalize well to unseen data.

Keras and TensorFlow have their own tools for preprocessing and evaluation, but Scikit-learn offers more utilities for tasks like feature scaling, data splitting, and model comparison. Its simplicity and compatibility with other machine learning models made it the best choice for preprocessing and evaluation.

6.2.4 Imbalanced-learn – Handling Class Imbalance with SMOTE

Imbalanced-learn is a Python library that provides tools for handling class imbalance in datasets, a common issue in intrusion detection systems. SMOTE (Synthetic Minority Over-sampling Technique) was used in this project to balance the class distribution in the training data. SMOTE generates synthetic samples for the minority class (attack traffic) by creating new, synthetic instances that are like existing minority class samples. This helps in reducing bias towards the majority class. It was applied to the training data before fitting the LightGBM classifier. This ensured that the model did not become biased towards some classes and could effectively learn to detect minority class attacks.

While other libraries like Scikit-learn provide basic methods for dealing with imbalanced data, Imbalanced-learn offers SMOTE as a more effective solution for oversampling the minority class. Other methods like undersampling were not used in this system, as they risk losing valuable data. SMOTE ensures that the model has a balanced representation of attack traffic during training.

6.2.5 Joblib – Model Persistence and Serialization

Joblib [24] was used for saving and loading models, scalers, and encoders to integrate them into the intrusion detection system.

Key features

- **Model Persistence**: The trained models (Autoencoder and LightGBM), as well as scalers and encoders, were saved using joblib. This ensures that the models can be reloaded and used for inference without retraining.
- **Saving/Loading Models**: The serialization mechanism of Joblib enables efficient storage of models together with their preprocessing steps which include scalers for feature normalization and encoders for categorical variables.

While Pickle serves as another Python library for serialization, Joblib outperforms it when handling big numerical data objects including trained models because of its optimized compression and quick read/write features. The library excels at saving machine learning models including LightGBM and Autoencoder for future inference purposes.

6.2.6 Kali Linux – Attack Simulation Tools

Kali Linux functions as a specialized penetration testing engine, which serves as a primary tool for cybersecurity research and attack simulation. It contains multiple tools which allow users to simulate different network attacks and vulnerability exploits to create a realistic attack environment. The libraries and commands in Kali served as

essential components for producing attacks against the victim machine to test the IDS capabilities.

6.2.6.1 Nmap – Network Scanning and Reconnaissance

Nmap [25] is a known tool for network exploitation and security auditing. It is mostly used for scanning networks, in order to discover open ports, active hosts and services, making it an essential tool for reconnaissance and network mapping.

Key features

- **Network Discovery**: Nmap allows fast and efficient network discovery, enabling the identification of devices and services running on the network. This is needed for detecting network scanning behaviors that often are before the actual attack.
- **Port Scanning**: It supports port scanning techniques, such as SYN scans and TCP scans. These are valuable for detecting the open ports so that the attacker can focus.
- **Stealth Mode**: Nmap offers features like stealth scanning, where scans are designed to evade detection by firewalls or IDSs, mimicking real-world attacking behavior (if not already used in real time attacks).

Nmap was used in this thesis to simulate reconnaissance attacks and port scanning, common precursor activities in cyberattacks.

6.2.6.2 Metasploit – Exploitation Framework

Metasploit [26] is an open-source framework used for developing and executing exploit code on remote targets. It is widely used for penetration testing and exploit development.

Key features

- **Exploit Development**: Metasploit allows the creation and execution of custom exploits against known vulnerabilities in the target systems. This capability was used to test the IDS' ability to detect attacks that exploit software vulnerabilities.
- **Payload Delivery**: Metasploit supports multiple types of payloads, such as reverse shells, to gain unauthorized access to a system.
- **Database Integration**: It can integrate with databases of known vulnerabilities, such as CVE (Common Vulnerabilities and Exposures), to simulate attacks based on the latest threats.
- **Automated Attack Execution**: It provides tools for automating the execution of multiple attack types.

Metasploit was used in order to simulate exploit-based attacks, where the attacker takes advantage of known vulnerabilities to gain unauthorized access to the victim machine.

6.2.6.3 Hydra – Brute Force Attacks

Hydra [27] is a popular tool used for brute-force attacks on various network services, including SSH, FTP and HTTP. Hydra is used to test the strengths of authentication systems by attempting to guess passwords using dictionary-based or brute-force methods.

Key features

- **Parallelized attacks**: This tool allows parallel brute-force attempts, by trying many password combinations at once, making it effective for cracking weak passwords in a short amount of time.
- **Support for multiple protocols**: Hydra supports a wide range of network services for brute-forcing.
- **Stealth Attacks**: Hydra includes features that allow stealthy attacks. This way it can evade detection of many IDS systems.

Hydra was used to simulate brute-force and credential-stuffing attacks, testing the IDS' ability to detect high-volume, rapid login attempts targeting network systems.

6.3 VIRTUAL MACHINE SETUP

The virtual machine setup is an important part of this thesis, as it simulates normal traffic and attack traffic in a controlled and isolated environment. By using VMs, the system can be tested in a flexible, reproductible and cost-effective way without the need for physical hardware. The two Virtual machines used in this setup were Ubuntu Linux (victim machine) and Kali Linux (attacker), which allowed us to simulate both benign and malicious traffic patterns without the need of an external network interface card.

6.3.1 Virtualization Technology

The virtual machines were set up using VMware, one of the most used platforms that host VMs. It is easy for creation, management and configuration of virtual environments that simulate real-world network setups. VMware provides support for network configurations that are essential for isolating virtual machines. This way it allows them to communicate with each other while ensuring no external interference. These tools also provide flexibility in resource allocation (CPU, memory) which is necessary for managing real-time traffic.

6.3.2 Network Configuration and Virtual Network Setup

For the effective simulation of network traffic and attack scenarios, a carefully designed network configuration was required. This section describes the virtual machine connection and how the network traffic between the victim machine and the attacker was captured, monitored and processed for real-time intrusion detection. The network

setup provided a controlled space to simulate both normal and malicious traffic which the Hybrid IDS could analyze.

Virtual Network Configuration

The two VMs (Ubuntu and Kali Linux) were connected via a private virtual network to simulate a local are network (LAN). This isolated network allowed a realistic simulation of network traffic, where both machines could communicate directly with each other but would be protected from external network. The virtual network setup ensured that the IDS would monitor the traffic between the two VMs without much "noise".

Both VMs were configured with static IPs to be sure that consistent monitoring would be easier to have. The IPs used for the victim and the attacker were 190.168.190.132 and 190.168.190.130 respectively. The victim machine's IP was used as target, while the attacker's was captured for analysis by the IDS.

Network Interface

The ens33 network interface was used for both the victim and attacker VMs to communicate with each other. This interface was configured in a host-only network mode in VMware meaning that the VMs can communicate with each other.

6.4 DATA COLLECTION AND ATTACK SIMULATION

The Hybrid IDS system depends on effective data collection and attack simulation to detect different network intrusions. This section explains the steps for network traffic collection and attack simulation execution and IDS testing for real-world and simulated attack scenarios.

6.4.1 Data Collection Process

Network traffic is continuously captured and analyzed for anomalies. The process starts with packet sniffing, followed by the aggregation of the packets into flows. These flows are analyzed for features that indicate potential attacks.

6.4.1.1 Network Flows

A network flow is a sequence of packets between a source and a destination that share some key attributes. The flow is recognized by the 5-tuple:

- **Source IP:** The IP of the machine which sends the packet.
- **Destination IP**: The IP of the packet receiver.

- **Source Port** The port number on the source machine from which the connection is initiated.
- **Destination Port**: The port number of the destination machine. It also represents the service or application that is targeted by the connection.
- **Protocol**: It defines the communication protocol between the two machines.

This 5-tuple represents a unique connection between two endpoints of the network. A flow represents packets between these two endpoints under the same 5-tuple during a single connection or communication.

6.4.1.2 Flow-based Data Collection

The system aggregates the captured packets into flows based on the 5-tuple. This aggregation consolidates packet-level data into more abstracted flow metrics that can be easier to analyze for anomalies.

The flow-based data collection was selected because of the ability to give the IDS more efficiency, better anomaly detection and scalability. This way the volume of data that needs to be analyzed can be reduced. Instead of inspecting every packet individually, the system can now aggregate them into a single flow. Additionally, some flow-level features, such as duration, number of packets and byte volume are more consistent during normal operations, but they can change during an attack like Denial-of-Service (DoS attack). Lastly, flow-based IDS, scales easier for a large network (for example in an organization), where monitoring every single packet would be computationally expensive. This way it can maintain high performance when dealing with large amounts of traffic.

6.4.2 Attack Simulation

In addition to the datasets, Kali Linux VM was used to simulate real-world traffic, generating normal and malicious traffic, including various types of attacks that the IDS was designed to detect.

Attack Types Simulated

- **Denial-of-Service (DoS**): DoS attack overwhelms the victim system by sending large volumes of traffic. These attacks are typically detectable by a sharp increase in flow duration and packet count, which are abnormal during typical network operations.
- **Reconnaissance and Port Scanning**: It involves multiple attempts to connect to different ports on the victim machine to determine which services are running. The IDS detects reconnaissance activity by looking for multiple connections to sequential ports from the same source IP and unusually high connection attempts within a short time frame.

- **Exploits**: Exploit attempts often involve abnormal network behavior, such as excessive retries, unusual protocol usage, and out-of-sequence packets. These are identified by the system based on flow-level features such as state transitions and packet arrival times.
- **Brute Force**: In brute-force attacks the attacker systematically attempts all possible combinations of passwords, encryption keys or authentication credentials, until the correct one is found. It can be identified by analyzing patterns, like continues attempts without successful authentication.

6.5 DATA PREPROCESSING AND FEATURE ENGINEERING

The development of an Intrusion Detection System requires both data preprocessing and feature engineering. The following section explains the procedures for dataset preparation and feature extraction in real-time to achieve the proper input for training and detection, for the Autoencoder and the LightGBM Attack Classifier systems.

6.5.1 Data Cleaning and Handling Missing Values

Before applying any machine learning model, the hybrid dataset needed to be cleaned and prepared for the training. This included handling missing values, dealing with error data entries and ensuring the integrity of the dataset.

Replacing Invalid Entries: The dataset included invalid entries with "-", which converted to NaN (not a number). It was done using the replace() function to enable the proper data cleaning. The data consisted of network protocols also with values like `http`, `ftp` and `tcp` values that also were replaced with NaN. These entries were marked for special handling to ensure they could be correctly encoded for the training of both systems.

Filling Missing Data: After replacing the invalid entries, the missing values inside the dataset were filled with zeros using the fillna() function. This approach is a common technique for handling missing values in a large dataset, where just like in this case, removing rows or columns could result in a loss of valuable data. By filling the missing values ensures that the model is not biased towards the null values, allowing it to learn from the available data without compromising its generalizability.

6.5.2 Label Encoding and Feature Encoding

Since the dataset included categorial features such as protocol and state, label encoding was used to convert these into numeric labels. This transformation is much needed especially in Autoencoder which works well with numeric features instead of strings.

Protocol Encoding: The `proto` feature, which represents the transport protocol used in communication, was encoded with Scikit-learn's Label Encoder. This step

transformed the string values of the protocols with integers, making it possible to train the Autoencoder with numeric features. For protocols not explicitly listed, the category `other` was used. This ensured the proper consistent encoding and handled any unexpected protocols.

State Encoding: The `state` feature, which represents the state of the connection, was label encoded the same way. This feature represents different connection states in the network and encoding it helps the models to understand better the behavior of the communication.

Attack Label Encoding: LightGBM classifier requires numeric input to perform the multiclass classification. For this reason, the attack category in the datasets was also label encoded. It was converted into numeric labels, corresponding to different attack types.

6.5.3 Feature Scaling and Normalization

Since the dataset contained features with different scales (e.g packet counts, byte volumes, flow duration), feature scaling was necessary to standardize the data. This ensured that no feature dominated the others due to its larger scale, helping the machine learning system to train more efficiently.

MinMax Scaling

The MinMax Scaler from Scikit-learn was used to scale the features into a fixed range of [0,1], which is especially important for models like the Autoencoder, as they are sensitive to the scale of input features. Without proper scaling, certain features could dominate the learning process due to their larger numerical values, leading to biased results. Below is a visual representation of feature scaling. In **Figure 6.2** a visual representation of feature scaling is depicted.

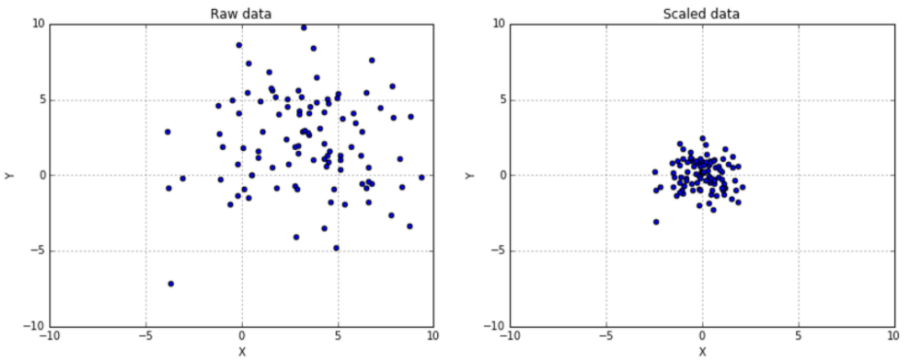

Figure 6.2: Visualized feature Scaling. [28]

This way Autoencoders benefit from feature scaling, as they rely on the ability to reconstruct inputs accurately. Large discrepancies in feature values can prevent the Autoencoder from learning the proper representations of data, thereby affecting anomaly detection performance.

6.5.4 Feature Engineering

The process of feature engineering stands essential for enhancing the performance of machine learning models. The system extracted new features from raw network traffic data which proved effective for detecting both anomalies and attacks.

Derived Flow Features

- **Packets per Duration (spkts_per_dur)**: This feature is calculated by dividing the total number of packets in the flow by the flow duration. It provides insight into how quickly packets are transmitted within a flow. Anomalous traffic, such as a DDoS attack, would likely show an increase in packet rate over time.
- **Bytes per Packet (bytes_per_pkt)**: This feature captures the average packet size within a flow. It is useful for detecting attacks that involve large payloads, such as data exfiltration or buffer overflow attacks.
- **Packets per Flow Duration (dpkts_per_dur)**: Like packets per duration, this feature calculates how many data packets are exchanged per unit of time in a flow. It is particularly useful in detecting DoS or DDoS attacks, which typically involve many packets being sent in a short time.

Connection Rate Metrics:

- **Rate**: This feature calculates the rate at which packets are transmitted between two endpoints. It can reveal spikes in traffic, especially during attack events like flooding attacks (e.g., DoS).
- **Sload and Dload**: These features represent the send load and receive load (in bits per second), respectively. They help in identifying unusual traffic flows where one side of the connection is overwhelmed with large amounts of data.

6.6 AUTOENCODER MODEL TRAINING

The Autoencoder functions as a vital element of the Hybrid IDS system which detects anomalies. The Autoencoder develops its ability to detect abnormal network behavior through training with normal traffic data. After training the model can detect anomalous traffic through reconstruction error analysis because high errors indicate the model failed to reconstruct the input correctly thus indicating possible attack activity.

Training the Autoencoder model involved several steps, from data preparation to model evaluation. The objective was to train the model solely on normal traffic and enable it to recognize anomalies in new, unseen data by detecting high reconstruction errors.

6.6.1 Training the model

Training Data

The model received its training data from normal network traffic within the hybrid dataset by using only flows that had a label of 0. The selection of normal samples served to instruct the Autoencoder about typical legitimate network patterns which enabled it to detect normal from anomalous traffic. The training process utilized selected key feature columns which contained essential network characteristics such as flow duration, packet count, byte volume and protocol usage. The model learned fundamental network traffic patterns through these attributes which would help detect potential attacks or anomalies later.

Training Configuration

- **Batch Size**: The model was trained with a batch size of 256, which is a good compromise between training time and model convergence.
- **Epochs**: The model was trained for 25-50 epochs with early stopping, where each epoch corresponds to one pass through the entire dataset. The training duration was chosen to balance performance with training efficiency.
- **Optimizer**: The Adam optimizer was used to adjust the weights during training. Adam is an adaptive optimization algorithm that combines the benefits of Momentum and RMSprop, providing fast convergence while minimizing the risk of getting stuck in local minima.
- **Learning Rate**: The learning rate was set to 0.001 to prevent overstepping during gradient updates, allowing the model to converge gradually.

Validation

K-Fold Cross-Validation was employed to assess the model's performance during training by dividing the dataset into 5 distinct subsets. In each fold, the model was trained on 4 subsets while being validated on the remaining one, which helped prevent overfitting to the training data. This technique ensures that the model's evaluation is not biased by a specific subset, providing a more reliable estimate of its performance. By using different combinations of training and validation sets, K-Fold Cross-Validation offers a better understanding of how well the model generalizes across various data points, ultimately enhancing its robustness and predictive accuracy.

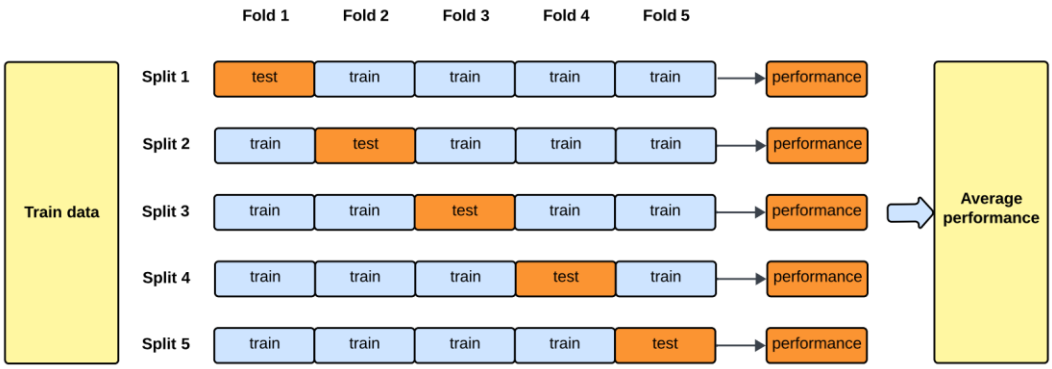

Figure 6.3: K-Fold Cross Validation. [29]

6.6.2 Challenges During Training and Solutions

Overfitting

The Autoencoder model implemented Dropout layers in both its encoder and decoder to prevent overfitting by randomly disabling specific neurons during training steps. The model learns to depend on different neuron subsets during each iteration through this regularization method which prevents it from relying on individual features and enhances its ability to predict new data. The model becomes more robust through this randomness which decreases its tendency to memorize training data and enhances its ability to process new network traffic.

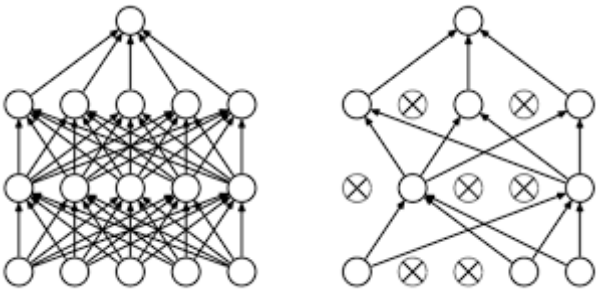

Figure 6.4: Dropout layers representation. [35]

Model Convergence

The Autoencoder training process showed slow convergence at first because it struggled to detect the intricate patterns found in network traffic. The model learned faster through optimized optimization by adjusting the learning rate which prevented it from overshooting the optimal solution. The model included early stopping mechanisms to stop training when validation loss stopped improving thus preventing overtraining. The

combination of learning rate adjustments with early stopping mechanisms allowed the model to achieve its best state without overfitting which resulted in better performance and generalization.

6.7 ATTACK CLASSIFIER (LIGHTGBM) MODEL TRAINING

The LightGBM (Light Gradient Boosting Machine) model was used for attack classification in the Hybrid Intrusion Detection System (IDS). LightGBM is a boosting algorithm that combines the output of multiple decision trees to create a robust model for both binary and multiclass classification tasks. In this system, LightGBM was employed for classifying network traffic flows into predefined attack categories (e.g., DoS, Exploits, Reconnaissance, etc.) after anomalies were detected by the Autoencoder.

6.7.1 Training the model

Model Training: The LightGBM model received training from the resampled dataset which used SMOTE to handle class imbalance and contained both normal and attack traffic flows. The training dataset contained network flows which received attack category labels from the UNSW-NB15 dataset. The training process used a broad selection of network traffic metrics which included flow duration and packet counts and byte volumes and protocol usage and other relevant characteristics. The model received protection against overfitting through early stopping which prevented it from continuing training after reaching satisfactory validation set performance. The model achieved better performance on the given data while maintaining its generalization ability through this approach.

Validation: After training the model, it was evaluated using the validation dataset, which included labeled network traffic featuring both normal and attack scenarios. The model's classification performance was assessed using a variety of evaluation metrics, including accuracy, precision, recall, F1-score, and the confusion matrix. These metrics provided a comprehensive view of how well the model identified both normal and anomalous traffic. Additionally, cross-validation was employed during the training phase to evaluate the model's stability and generalization. The dataset was split into training and validation sets using stratified sampling, ensuring that both sets maintained the same distribution of classes (normal and attack) for a more reliable and balanced evaluation.

6.7.2 Challenges During Training and Solutions

Handling Class Imbalance: The UNSW-NB15 dataset shows a major class imbalance because normal traffic instances far exceed attack traffic instances. The model probably preferred normal traffic because it was more abundant than attack traffic which resulted in poor detection of attack traffic. The model used SMOTE (Synthetic Minority Over-sampling Technique) to create artificial minority class instances for balancing the class distribution. The LightGBM model used class_weight='balanced' to train the model with equal attention to the minority class. The model received proper attention to rare attack patterns through this approach which enhanced its attack detection capabilities without developing bias toward the majority class.

Tuning Hyperparameters: The process of finding the best hyperparameters for LightGBM took a lot of time and required careful adjustment. The model is very sensitive to the hyperparameters such as learning rate, number of leaves, max depth and n_estimators and selecting wrong values could result in overfitting or underfitting. GridSearchCV was used to systematically search for the best combination of hyperparameters. This technique exhaustively searched through different hyperparameter values to identify the optimal parameters that balanced model complexity with training efficiency. The result was a well-generalized model that performed well in different data scenarios, ensuring robust performance without overfitting.

Overfitting: Overfitting is a common challenge when training on highly imbalanced datasets, especially with complex models like LightGBM, which have numerous hyperparameters. A model that becomes too complex may excel on the training data but struggle to generalize to unseen data, resulting in poor performance in real-world applications. To mitigate this risk, regularization techniques such as adjusting class weights and implementing early stopping were applied to prevent overfitting. Cross-validation was used to assess model stability and generalization, while pruning techniques helped reduce the model's complexity without sacrificing performance on the validation set. This combination of strategies ensured that the model remained both robust and efficient in real-world scenarios.

Multiclass Classification Complexity: Training a model for multiclass classification, where the goal is to classify various types of attacks (e.g., DoS, Exploits), is inherently more complex than binary classification. Each class requires the model to not only identify whether traffic is normal or anomalous but also to differentiate between multiple categories of attacks. To address this complexity, LightGBM's multiclass objective was employed, allowing the model to handle multiple classes effectively. Additionally, class-wise tuning was performed to optimize performance across all

categories, ensuring that the model could accurately distinguish between the different attack types. The classification threshold was also adjusted to balance precision and recall for each class, ensuring the model maintained robust performance across all attack types while minimizing false positives and negatives.

6.8 INTRUSION DETECTION SYSTEM

In this section, we describe how all the components of the Hybrid IDS are integrated into a unified system that allows for real-time anomaly detection, attack classification, and automated responses. The system incorporates the Autoencoder model for anomaly detection, the LightGBM classifier for attack classification, and a real-time response mechanism that blocks malicious IP addresses using iptables. Additionally, we will discuss how we use joblib for saving and loading models and encoders, which makes the system efficient and easy to manage.

The system captures and processes network traffic in real-time, continuously detecting anomalies and classifying attacks as they occur. When the autoencoder system detects an anomaly, the corresponding network flow is passed to the LightGBM classifier for attack type classification. To ensure efficient model inference and facilitate real-time detection without the need for retraining, both the Autoencoder and LightGBM models are loaded at runtime using joblib [24]. This approach allows the models to operate quickly and seamlessly, enabling the system to identify threats in real-time while maintaining optimal performance. The data preprocessing and feature transformation is the same as both systems use, so integration is possible to make.

6.8.1 Automated IP Blocking/Unblocking Mechanism

IP Blocking: Once an attack is detected by the Autoencoder and LightGBM classifier, the system automatically blocks the attacker's IP using iptables, a Linux tool for packet filtering. This IP blocking mechanism helps prevent further malicious traffic from reaching the victim machine, mitigating the attack in real time. The system executes a command, which adds a rule to the iptables firewall, instructing it to drop all incoming traffic from the specified attacker's IP address. This approach ensures that the attacker's malicious activity is immediately halted, protecting the system from further harm.

Timed Unblocking: The system includes a time-based unblocking mechanism to stop blocking legitimate users permanently. The system monitors the unblocking time (e.g., 100 seconds) for blocked attacker IPs before automatically restoring access to the IP after the specified duration ends. The ip_unblock_times dictionary functions as a tracking system for unblocking times of each IP address. The system removes the block after the blocking period ends it unblocks the IP and deletes the rule from the iptables firewall thus enabling the previously blocked IP to send traffic. The system design prevents legitimate users from enduring prolonged restrictions.

Flow Cleanup: After blocking an attacker's IP, the system removes all existing flows associated with that IP from the current session. This step ensures that any malicious traffic related to the blocked IP is no longer processed. By eliminating these malicious flows, the system not only prevents unnecessary processing but also reduces the overall load on the IDS, allowing it to focus on legitimate traffic and other potential threats.

6.8.2 Integration of Models and Real-time Decision-Making Flow

The system operates in real-time to detect anomalies and classify attacks through continuous network traffic capture and processing. The LightGBM classifier receives the flow after anomaly detection to determine the exact attack type. The system maintains real-time detection capabilities through runtime loading of both Autoencoder and LightGBM models using joblib for efficient model inference. If a malicious flow is identified, the system triggers the IP blocking mechanism, automatically isolating the attacker from the network to prevent further damage or intrusion. This dual-layer detection and response system helps maintain network security in real-time by swiftly identifying and neutralizing threats.

6.9 Challenges and Limitations

The Hybrid IDS system showed excellent performance in detecting and classifying attacks, yet several challenges and limitations were encountered during both the development and real-time testing phases. The system encountered various obstacles including data problems and model performance issues and scalability limitations and the fundamental constraints of IDS systems when operating in real-time network environments. The following section presents the main obstacles encountered during implementation together with their solutions and the current system's limitations.

6.9.1 Data Quality and Availability

Machine learning systems achieve their performance through data quality and availability. The training and evaluation models of intrusion detection systems require data that represents actual network traffic while including diverse attack scenarios. The following section examines the dataset quality issues and labeled data availability problems which affected LightGBMs system training and evaluation processes.

Data Quality: The performance of any machine learning-based system is largely dependent on the quality of the data used for training. In this system, while the UNSW-NB15 dataset is comprehensive, it contains some noise and inaccuracies that affected the model's performance. To address this issue, various data cleaning techniques were employed, such as replacing invalid entries (e.g., "-") with NaN values and filling missing data with zeros. These steps helped standardize the dataset and prevent the model from being misled by incomplete or erroneous entries.

Class Imbalance: The main problem in intrusion detection systems arises from class imbalance because normal traffic instances are way more than the attack ones. This problem was solved by using the Autoencoder system for basic detection, because it is trained only by using the normal instances of the data. Also, for the classifier, its training process used SMOTE to increase the number of minority classes to achieve data balance.

6.9.2 Model Complexity and Overfitting

One of the key challenges in machine learning, particularly in complex systems like an intrusion detection system (IDS), is managing model complexity and avoiding overfitting. In this section, we will explore the difficulties encountered in controlling model complexity during the training of both the Autoencoder and LightGBM models, and the strategies employed to mitigate overfitting while ensuring that the system performs well in real-world scenarios.

Overfitting: One of the key challenges in machine learning, particularly in complex systems like an intrusion detection system (IDS), is managing model complexity and avoiding overfitting. Overfitting occurs when a model learns the noise or irrelevant patterns in the training data, leading to poor generalization on new, unseen data. In this section, we will explore the difficulties encountered in controlling model complexity during the training of both the **Autoencoder** and **LightGBM** models, and the strategies employed to mitigate overfitting while ensuring that the system performs well in real-world scenarios. Overfitting is a common challenge when training machine learning models, particularly with highly complex datasets. In the case of the Autoencoder and LightGBM, there was a risk that the models would perform well on the training data but poorly on unseen data, leading to a lack of generalization. To mitigate this, several strategies were employed: For the Autoencoder, dropout layers were introduced in the architecture to randomly drop neurons during training, preventing the model from learning overly complex relationships that might not generalize well to new data. For LightGBM, early stopping was implemented, halting training if the model's performance on the validation set did not improve after a certain number of epochs. This helped avoid unnecessary overfitting during extended training. Lastly, K-fold cross-validation was used during the training process to assess the model’s generalization performance across different subsets of the data, providing a more robust evaluation of how the model would perform on unseen data.

Hyperparameter Tuning: Model performance optimization depends heavily on hyperparameter tuning yet this process consumes significant time and computational resources. The process of optimizing LightGBM parameters including learning rate and

number of leaves and max depth demanded extensive testing to discover the best possible values. The model received efficient tuning through the implementation of GridSearchCV. The technique performed automated hyperparameter tuning by testing different parameter combinations in a systematic way. The method conducted an extensive search of hyperparameters to find the best combination which optimized both model accuracy and computational performance.

6.9.3 Scalability and Real-time Processing

As the Hybrid IDS system is designed for real-time network traffic analysis, ensuring that it can handle high volumes of data while maintaining low latency is essential. Scalability becomes a critical factor when the system is deployed in larger network environments with increasing traffic loads. This section discusses the challenges related to scalability, particularly the system's ability to process real-time traffic efficiently, while maintaining high detection accuracy and response times in varying network conditions.

High-Volume Network Traffic: The Hybrid IDS system was designed to process network traffic in real time, but high traffic volumes can strain the system’s ability to maintain low latency and high throughput. As the network scales, the amount of data to process increases, which may result in delays or slowdowns in the system's response time. To mitigate this, flow-based detection was used instead of packet-based analysis, reducing the amount of data that needs to be processed at once. Also feature extraction played a big role in managing processing time, making it possible for only the most relevant features to be used in training. Additionally, multi-threading was used for packet-sniffing and flow-processing to enable parallel operations and reduce time taken for each detection cycle.

Real-Time Detection: Ensuring real-time detection with low latency was essential for the IDS to be effective. However, processing traffic in real-time while maintaining high-model accuracy and classification performance posed a significant challenge, especially when the system was deployed in high-traffic networks. The Autoencoder and LightGBM models were optimized for low-latency inference. The preprocessing steps of feature scaling and label encoding were made efficient by caching the scalers and encoders using joblib so they were ready to be applied during real-time detection without any delays.

6.9.4 False Positives and False Negatives

The main challenge in intrusion detection systems (IDS) is to detect attacks while minimizing false positives (normal traffic incorrectly flagged as attacks) and false

negatives (attacks missed by the system). False positives can overwhelm administrators with alerts, while false negatives leave the system vulnerable to undetected attacks. This section discusses the challenges faced in managing false positive and false negative rates in the Hybrid IDS system, and the strategies used to mitigate these issues while ensuring effective and efficient detection.

False Positives: An important concern in any anomaly detection system is the trade-off between detection sensitivity and false positives. In this system, there was a tendency to flag normal behavior as anomalies due to slight deviations in traffic patterns, leading to unnecessary alerts. To address this, the Autoencoder utilized dynamic thresholding, which allowed the system to adapt to varying network conditions and adjust its sensitivity based on the current traffic behavior. By doing so, the system was able to reduce the number of false positives, particularly during periods of normal traffic fluctuations, ensuring that only significant anomalies were flagged as potential threats.

False Negatives: The biggest challenge was false negatives, where the system missed detecting certain attacks. Some attacks, especially low-rate attacks or sophisticated attack techniques, might not show enough deviation from normal behavior to be flagged as anomalies. To address this, the system used unsupervised anomaly detection. The Autoencoder detects anomalies that diverge from the normal pattern easier than any other detection system because of its architecture. Post-processing techniques like anomaly aggregation were employed to ensure that even subtle attacks did not go undetected.

6.9.5 Limitations

The Hybrid IDS system demonstrates various achievements and innovative aspects, yet it faces multiple built-in restrictions which could affect its operational effectiveness or adaptability in specific real-world applications. The system faces performance limitations because of training data quality and model complexity and network environment dynamics. The following section examines the main system development and real-time deployment challenges which future system versions should address.

Data dependency: The system heavily depends on the quality of labeled datasets for training. While the hybrid dataset is comprehensive, it may not cover all possible attack vectors, especially newer attack types or evolving threats. As such, the system's performance on unknown or unseen attacks is limited by the training data available. The use of the Autoencoder system was crucial to solve this problem. The fact that it is training only by using the benign network traffic makes it sufficient to understand and detect even zero-day attacks.

Resource Consumption: The real-time detection process, especially with multiple models (like Autoencoder and LightGBM), requires considerable computational resources. While the system was designed to run efficiently, processing large volumes of traffic at high speeds may still demand significant computational power. The use of model optimization techniques, such as quantization or pruning for LightGBM, and leveraging GPU acceleration for Autoencoders, could help reduce resource consumption and speed up inference times.

6.10 FUTURE WORK

While the Hybrid IDS system successfully integrates anomaly detection using Autoencoders and attack classification with LightGBM, there is always room for improvement and adaptation to new challenges. This section outlines some of the key areas for future work to enhance the system's capabilities, address its current limitations, and expand its functionality in real-world environments.

6.10.1 Expansion of Dataset

The hybrid dataset provided valuable labeled data for training and evaluation, but it has limitations in terms of attack types and network scenarios. Future work could involve incorporating more diverse and updated datasets, such as CICIDS or KDDCup99, or real-world traffic data from honeypots. Integrating online learning or continual learning techniques would allow the system to learn from new data over time, improving detection accuracy and adaptability to emerging attack vectors.

To address the issue of class imbalance, SMOTE was used during training to generate synthetic samples for the minority class. However, future research could explore generative models like GANs (Generative Adversarial Networks) to create more diverse and realistic attack traffic samples for training. This could enhance the system's robustness in detecting rare or novel attack types.

6.10.2 Enhancing Model Performance

The Autoencoder would benefit from deep learning architectures including Variational Autoencoders (VAEs) and Convolutional Autoencoders to enhance their ability to learn complex normal traffic patterns. The Autoencoder would detect subtle anomalies in complex network behaviors more effectively with these improvements. Future research should develop ensemble models that unite Autoencoders with LightGBM and other machine learning methods including Random Forests and Neural Networks. The system would achieve better detection accuracy and lower false positive rates through the combination of multiple model outputs. Ensemble techniques that use stacking to train different models on separate feature sets would be particularly effective for enhancing multiclass classification performance.

6.10.3 Real-Time and Scalable Deployment

For larger-scale networks or environments with high data throughput, edge computing could be leveraged to distribute the processing load closer to the source of data (e.g., on routers or network gateways). This would reduce the reliance on central servers and decrease the latency associated with real-time detection. Future work could explore cloud-based deployment for scalable and distributed network monitoring. By integrating the system with cloud platforms like AWS or Azure, the IDS could be scaled to handle larger volumes of traffic from multiple network segments simultaneously.

CHAPTER 7: EVALUATION AND RESULTS

The evaluation of the Hybrid Intrusion Detection System (IDS) in this chapter combines Autoencoder-based anomaly detection with LightGBM for attack classification. The evaluation system aims to determine how well the system detects network anomalies and identifies different attack types while responding to malicious activity in real time. The evaluation will use multiple performance metrics to assess the system including accuracy, precision, recall, F1-score, false positives and false negatives.

7.1 AUTOENCODER SYSTEM PERFORMANCE

The basic Autoencoder was trained exclusively on benign traffic and learned the intrinsic patterns of normal network activity. When exposed to anomalous data, it failed to reconstruct those patterns accurately, producing a high reconstruction error used as the anomaly signal.

7.1.1 Loss Function

The generalization capability and stability of the autoencoder model were evaluated through 5-fold cross-validation using only normal samples from the training set. The training and validation loss curves for each fold are shown in Figure X. The graph shows that all folds decrease their reconstruction loss (Mean Squared Error) consistently during the 25 epochs of training which proves the model learned normal traffic patterns successfully. The majority of folds reached a low and stable loss value which demonstrates the model's effective reconstruction of benign traffic.

The initial loss of Fold 1 was higher than the other folds indicating this subset contained more complex or varied normal class samples. The validation loss of this fold followed the same downward pattern until it reached the same loss range as the other folds. The model demonstrated robustness because it did not overfit and maintained its performance across different data partitions.

The model demonstrates good generalization capabilities because its training and validation loss curves in **Figure 7.1** match closely which indicates resistance to overfitting thus making it appropriate for future anomaly detection through reconstruction error.

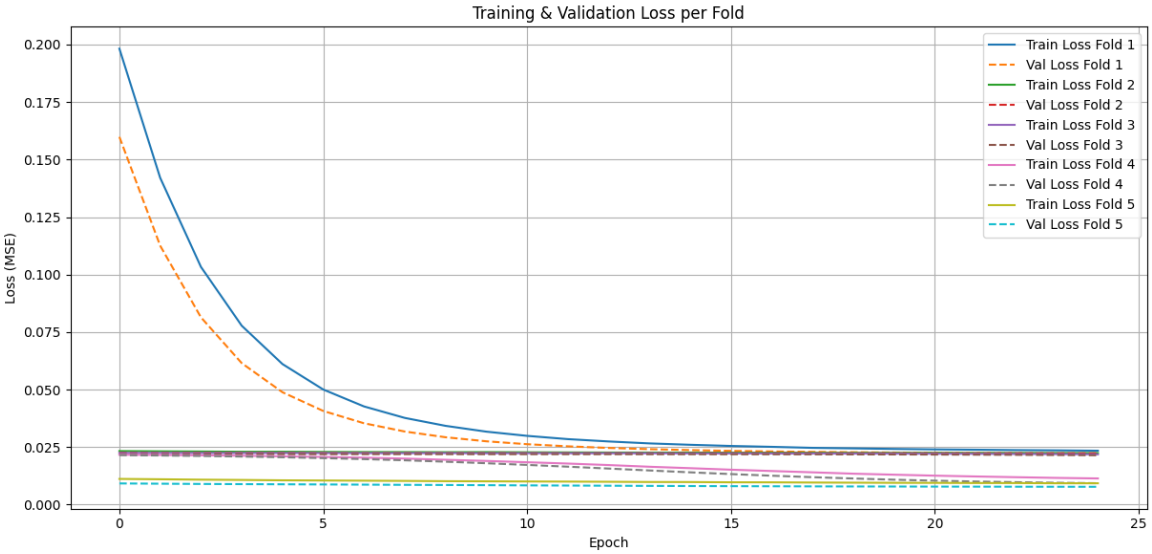

Figure 7.1: Autoencoder Training & Validation Loss.

7.1.2 Classification Report

- **Overall Accuracy:** 80%
- **Attack Precision:** 77%
- **Normal Traffic Precision:** 78%

Class	Precision	Recall	F1-Score
0 (Normal)	78%	76%	77%
1 (Attack)	77%	79%	77%

Table 7.1: Autoencoder Classification Report.

Emphasis was placed on minimizing false negatives, accepting more false positives to ensure fewer attacks bypass detection.

7.1.3 Confusion Matrix

The Confusion Matrix is a table used to evaluate the performance of a classifier. It shows how many of the predictions made by the model were correct or incorrect, broken down by each class, providing insights into the types of errors the model is making.

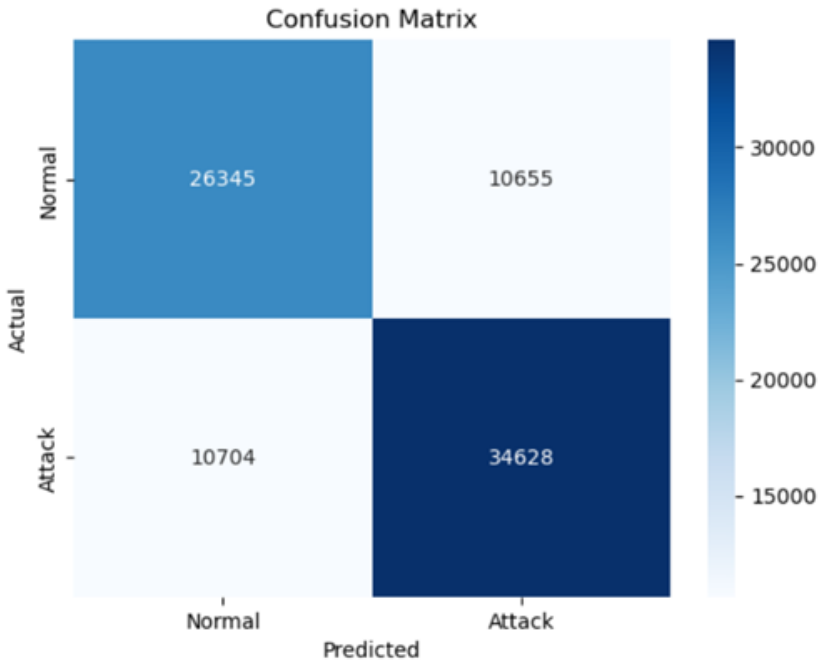

Figure 7.2: Autoencoder System Confusion Matrix.

The confusion matrix in **Figure 7.2** shows the classification results of the anomaly detection system using the reconstruction error threshold learned from the trained autoencoder. The model was tested on a dataset that contains both normal and attack samples. The model correctly identified 26,345 normal samples (True Negatives) and 34,628 attack samples (True Positives), which shows that the model is very good at distinguishing between normal and malicious traffic. However, 10,704 attacks were incorrectly classified as normal (False Negatives), and 10,655 normal samples were misclassified as attacks (False Positives).

These results show a balanced detection pattern, where the model has high true positive and true negative rates. However, the presence of false positives and false negatives shows the inherent trade-off in threshold-based anomaly detection systems.

7.1.4 Insights

- This model was lightweight and efficient, making it ideal for deployment in real-time systems with limited computational resources.
- The system proved effective at detecting anomalies which made it suitable for detecting both known and unknown zero-day attacks.
- The false positive rate was preferred to be relatively high, instead of having higher false negative, but it could lead to alert fatigue in security operations.

7.2 LIGHTGBM CLASSIFIER SYSTEM PERFORMANCE

LightGBM (Light Gradient Boosting Machine) was deployed as a supervised multiclass classifier to detect and categorize diverse cyberattacks using the hybrid dataset. Given the dataset's inherent class imbalance, the SMOTE (Synthetic Minority Over-sampling Technique) algorithm was applied to generate synthetic samples for minority attack classes, enhancing model generalization and minority-class recognition.

7.2.1 Classification Report

Before SMOTE

- **Overall Accuracy:** 87.49%

Class	Precision	Recall	F1-Score	Samples
Exploits	90%	87%	83%	11.132
Generic	100%	100%	100%	18.871
Normal	94%	90%	90%	37.000
DoS	70%	65%	70%	4089
Fuzzers	82%	81%	79%	6062
Backdoor	40%	40%	40%	583
Worms	8%	27%	21%	44
Shellcode	11%	37%	17%	378
Reconnaissance	78%	57%	60%	3496

Table 7.2: LightGBM Classification Report (Before SMOTE).

The classification report before applying SMOTE shows an overall accuracy of 87.49%. The model performs well on Generic traffic, with perfect precision and recall, and on Normal traffic, with 94% precision and 90% recall. However, performance drops on minority classes such as Worms and Shellcode, where precision and recall are significantly low. These results highlight the challenge of class imbalance, where the

model struggles to correctly identify rare attack types, underscoring the need for techniques like SMOTE to address this issue.

After SMOTE

- **Overall Accuracy:** 89.6%

Class	Precision	Recall	F1-Score	Samples
Exploits	91%	89%	90%	11.132
Generic	100%	100%	100%	18.871
Normal	94%	90%	90%	37.000
DoS	74%	72%	73%	4089
Fuzzers	85%	84%	84%	6062
Backdoor	65%	62%	63%	583
Worms	47%	50%	48%	44
Shellcode	51%	59%	54%	378
Reconnaissance	83%	72%	77%	3496

Table 7.3: LightGBM Classification Report (After SMOTE).

After applying SMOTE to address class imbalance, the model's overall accuracy increased to 89.6%. Significant improvements were seen in the detection of minority classes like Worms and Shellcode, with precision and recall rising to 47% and 50%, respectively, for Worms, and 51% precision and 59% recall for Shellcode. While the model's performance for rare attack types has improved, there is still room for further enhancement, especially in detecting smaller classes like Backdoor and Worms. These results demonstrate the effectiveness of SMOTE in improving the detection of underrepresented attack types.

7.2.2 Insights

- **SMOTE significantly improves recall** for rare attack classes, reducing class imbalance effects.
- **Precision remains high for generic and normal traffic** but is still lower for underrepresented classes, requiring further optimization.

7.3 DUAL AUTOENCODER (ANOMALY DAE) SYSTEM PERFORMANCE

AnomalyDAE combines two distinct Autoencoders:

- One learns structural relationships in network data (Structure Autoencoder).
- The other captures attribute-level features (Attribute Autoencoder).

This architecture was inspired by research on attributed networks, offering a more nuanced detection of anomalies through richer latent representations.

7.3.1 Loss Function

The following graph depicts the training and validation loss (Mean Squared Error) of a Dual Autoencoder model over 25 epochs during 5-fold cross-validation. Initially, both the training and validation losses decrease rapidly, indicating the model is learning effectively. The training loss continues to decline throughout the epochs, while the validation loss stabilizes after a few epochs, suggesting the model is generalizing well to unseen data. This steady decrease in both losses without significant overfitting (i.e., the gap between the training and validation loss does not grow substantially) indicates that the model is successfully learning the underlying patterns of the data while maintaining its ability to generalize. **Figure 7.3** shows the average training loss during Dual Autoencoder's training.

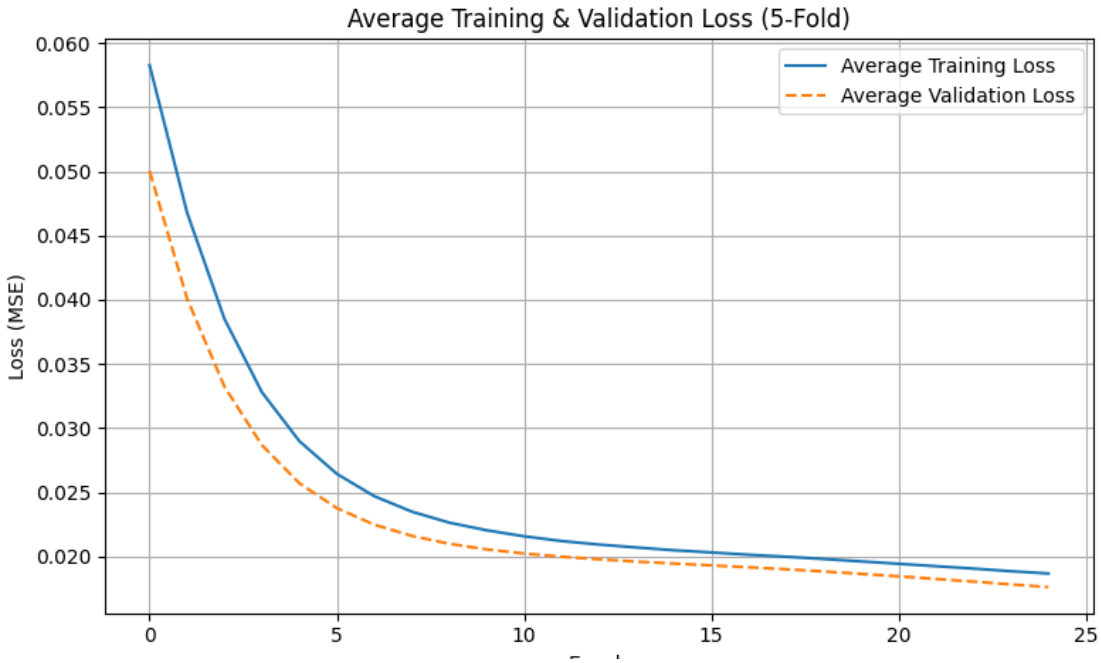

Figure 7.3: Dual Autoencoder Training Loss.

7.3.2 Classification Report

- **Overall Accuracy:** 85%
- **Attack Precision:** 84%
- **Normal Precision:** 78%

Class	Precision	Recall	F1-Score
0 (Normal)	78%	77%	77%
1 (Attack)	84%	82%	80%

Table 7.4: Dual Autoencoder (Anomaly DAE) Classification Report.

The **table 7.4** shows the classification performance of a Dual Autoencoder model for attack detection, with precision, recall, and F1-score for both the "Normal" (0) and "Attack" (1) classes. The model performs well, with an 84% precision and 82% recall for the "Attack" class, resulting in an F1-score of 80%. For the "Normal" class, the model achieves a precision of 78%, recall of 77%, and an F1-score of 77%. These results show that the Dual Autoencoder performs reasonably well in distinguishing between normal and attack behaviors, with better performance in identifying attacks compared to normal traffic.

7.3.3 Confusion Matrix

The confusion matrix shown in **Figure 7.4** displays the performance of the Dual Autoencoder model. It reveals how well the model differentiates between "Normal" and "Attack" traffic. The top-left cell indicates that 30,642 "Normal" instances were

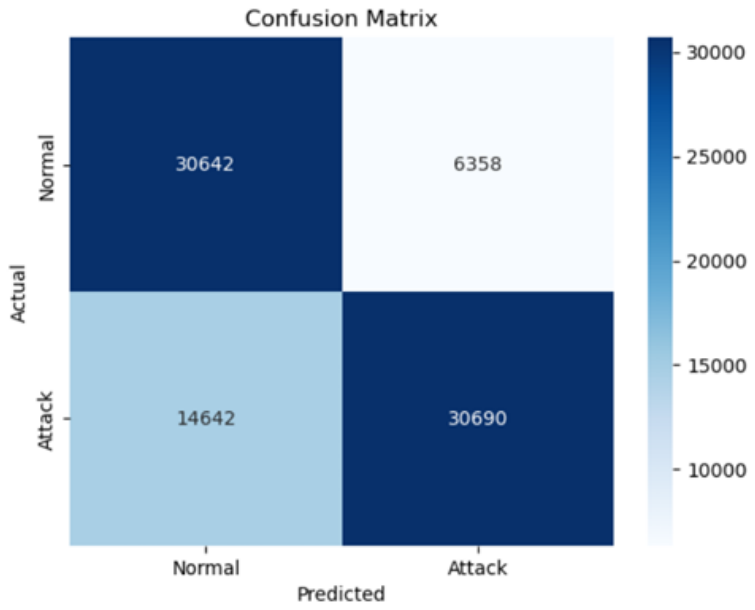

Figure 7.4: Dual Autoencoder (Anomaly DAE) Confusion Matrix

correctly identified as "Normal," while the top-right cell shows 6,358 "Normal" instances were incorrectly classified as "Attack." In the bottom-left, 14,642 "Attack" instances were wrongly identified as "Normal," and the bottom-right cell reveals that 30,690 "Attack" instances were correctly identified. While the model exhibits strong performance with a high number of correct classifications, the presence of false positives and false negatives suggests room for improvement, especially in fine-tuning the model to reduce misclassifications in real-world network traffic.

Despite the Dual Autoencoder's promising performance, the single Autoencoder was ultimately preferred. The single Autoencoder, while having slightly lower accuracy and performance metrics compared to the Dual Autoencoder, was faster and more efficient in real-time settings. Since the real-time nature of the application was a priority, and the accuracy and performance differences were not significant, the single Autoencoder provided a better trade-off between speed and detection performance, making it the optimal choice for deployment.

7.4 INTRUSION DETECTION SYSTEM PERFORMANCE

By integrating Autoencoder anomaly detection and LightGBM attack classification, the system was able to detect and classify both known and unknown attack types.

7.4.1 Results

The system achieved a total detection accuracy of 87% which demonstrated its effective ability to identify potential threats. The system achieved 76% accuracy in attack classification which demonstrated its capability to identify various attack types. The system achieved an F1-score of 0.91 which demonstrates its ability to strike a good balance between precision and recall and its effectiveness in both attack detection and false positive reduction. The system demonstrates a decent performance for both detection and classification tasks based on these results.

7.4.2 Real-time Testing Performance

The system's real-time performance was evaluated by processing network traffic as it occurred. The latency for processing and detecting anomalous flows was measured, with the average detection time per flow being less than 0.5 seconds. This rapid detection time makes the system well-suited for deployment in high-throughput networks, where quick identification of threats is essential to maintaining security and minimizing potential damage.

7.5 SYSTEM PROTOTYPE OPERATION

Figure 7.5, 7.6 and 7.7 are detailed examples of attacks and detections from the Intrusion Detection System (IDS). On the left side, we see the Ubuntu machine, which is the legitimate server, and on the right side, the attacking Kali Linux machine attempting to initiate the attack. The images illustrate the various stages of the attack attempt from the Kali Linux machine, followed by the disruption caused by the IP block. This block prevents the attacker from reaching the server, effectively neutralizing the threat in real-time and demonstrating the IDS's ability to protect the system by halting malicious activity.

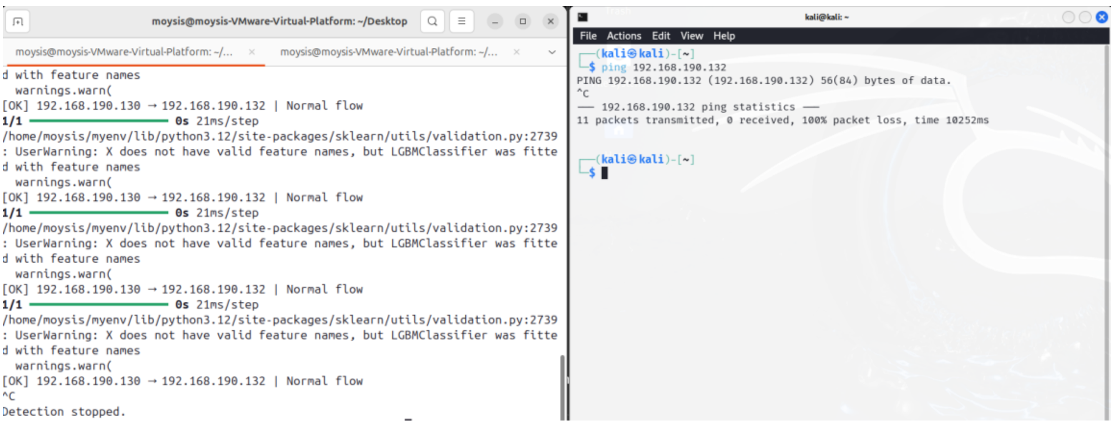

Figure 7.5: Normal Flow Example.

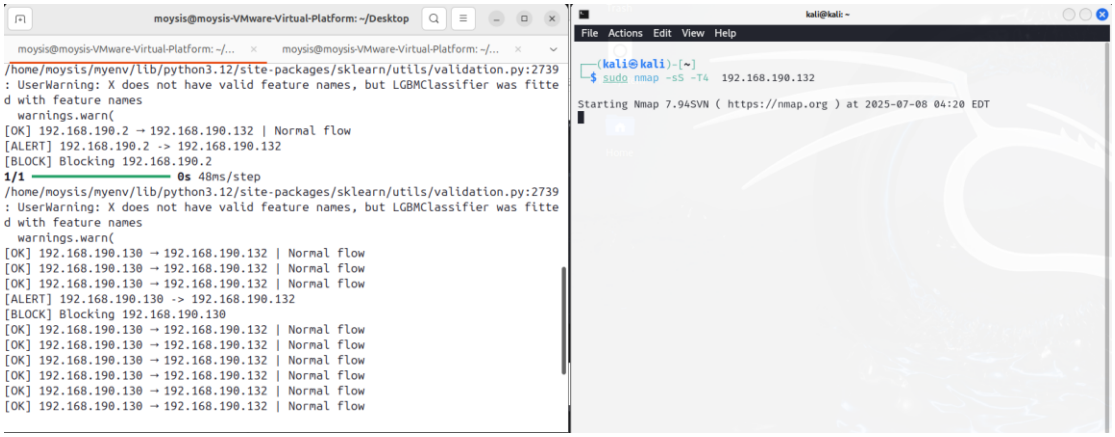

Figure 7.6: Stealth Reconnaissance Example.

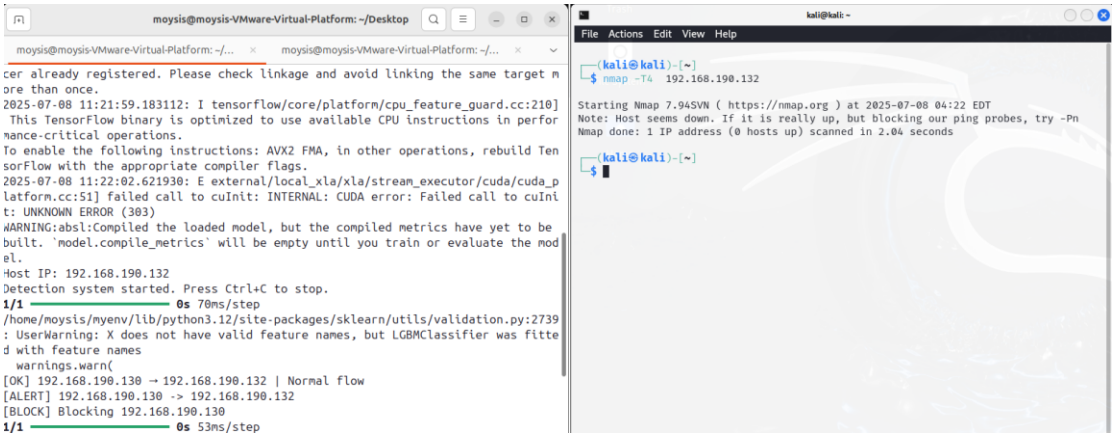

Figure 7.7: Reconnaissance Example.

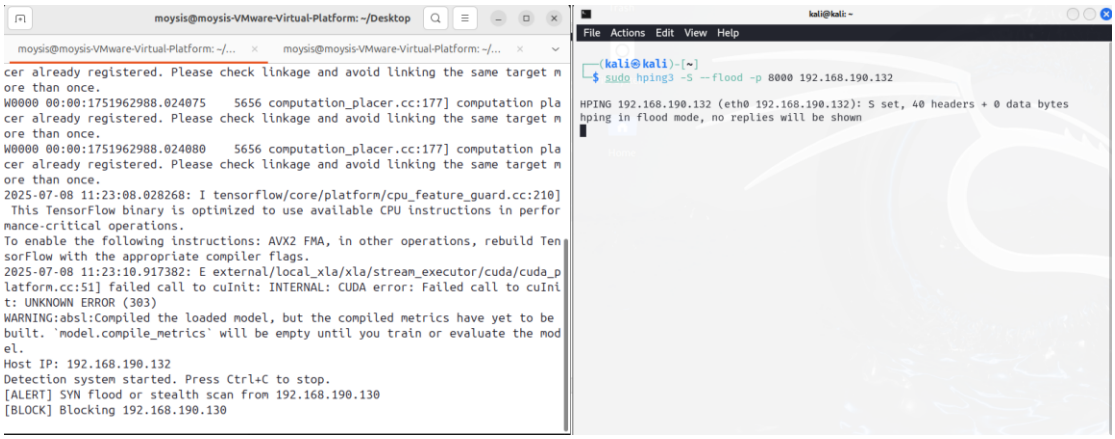

Figure 7.8: DoS (hping3 flood) Example.

CHAPTER 8: CONCLUSIONS

The main objective of this research involved creating and assessing a Hybrid Intrusion Detection System (IDS) which unites Autoencoder-based anomaly detection with LightGBM-based attack classification. The two models combined into a powerful system which detected both new and recognized attacks in real-time network traffic. The system achieved excellent results through thorough testing because it correctly identified normal and malicious network traffic while producing minimal incorrect classifications.

The system demonstrates its main advantage through its ability to adjust to new security threats. The Autoencoder learned normal traffic patterns to detect anomalies without needing prior knowledge of specific attack signatures. The system proved highly successful at detecting zero-day attacks and new attack types because it lacked dependence on signature-based systems. The system's performance heavily relied on the quality of training data, especially when using labeled datasets because these datasets might not include all possible attack scenarios.

The Autoencoder detected an anomaly which triggered the flow to proceed to the LightGBM classifier for attack type classification between DoS, Fuzzers, Generic, Shellcode, Worms, Reconnaissance, Backdoors and Exploits. The system used SMOTE to address class imbalance while applying class weighting to LightGBM which resulted in high accuracy detection of both common and rare attacks with balanced precision and recall.

The system achieved real-time success because its detection and classification operations finished within sub-second timeframes. The system is ready for deployment in high-speed networks because it provides fast attack response times. The system requires future development to address false positive occurrences and model retraining because network traffic patterns change continuously.

Throughout the course of this research, several insights emerged that were not obvious at the outset but became clear during the development and evaluation process:

The role of **data preprocessing and dataset quality** became evident. Even the most advanced models underperform if the input data is not properly cleaned, balanced and representative

Unsupervised learning is **not a shortcut**. While Autoencoders eliminate the need for labeled attack data, they require careful tuning, and their performance is highly sensitive to the definition of “normal” traffic.

Real-time performance is harder than theoretical performance. Models that perform well in offline evaluations can still struggle under real-time constraints due to latency, memory or throughput issues. Achieving sub-second performance required significant optimization.

Security is a moving target, not a fixed goal. One of the most surprising realizations was how quickly threat landscapes evolve. A system trained on today's attacks may fail tomorrow. This emphasized the importance of not only initial accuracy but also long-term maintainability, retraining mechanisms, and adaptive learning as core components of any security solution.

In conclusion, this work demonstrates that combining unsupervised anomaly detection with supervised attack classification offers a promising approach for network intrusion detection. While the Hybrid IDS has shown strong potential, addressing the challenges of data quality, model scalability, and false positives will be crucial for its deployment in real-world scenarios. Future research will focus on enhancing the system's adaptability, improving detection capabilities, and ensuring its scalability for large-scale network environments.

REFERENCES

[1] Statista, "Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2023, with forecasts from 2024 to 2028," 21 November 2024. [Online]. Available: https://www.statista.com/statistics/871513/worldwide-data-created. [Accessed 2025].

[2] S. Morgan, "Cybercrime To Cost The World $10.5 Trillion Annually By 2025," 17 August 2016. [Online]. Available: https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/. [Accessed 2025].

[3] NPR, "T-Mobile says data from 37 million customers was stolen," NPR, 20 January 2023. [Online]. Available: https://www.npr.org/2023/01/20/1150215382/t-mobile-data-37-million-customers-stolen. [Accessed 2025].

[4] N. Bisceglia, "MGM Resort Cyberattack: How Hackers Shattered Operations with $100M in Damages," TeamPassword, 25 February 2025. [Online]. Available: https://teampassword.com/blog/mgm-resort-cyberattack. [Accessed 2025].

[5] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks,* vol. 51, p. 3448–3470, 2007.

[6] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security,* vol. 28, no. 1-2, p. 18–28, 2009.

[7] IBM Security, "Cost of a Data Breach Report 2023," IBM, Armonk, NY, 2023.

[8] Symantec, "Internet Security Threat Report," Symantec Corporation, Mountain View, 2019.

[9] S. Park, "Unveiling AI Agent Vulnerabilities Part II: Code Execution," Trend Micro Inc., Tokyo, 2025.

[10] M. Sakurada and T. Yairi, "Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, ACM, 2014.

[11] Y. Mirsky, T. Doitshman, Y. Elovici and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online," NDSS Symposium, Negev, 2018.

[12] F. Haoyi, F. Zhang and Z. Li, "AnomalyDAE: Dual autoencoder for anomaly detection on attributed networks," ICASSP2020, Harbin, 2020.

[13] J. Zhang, Y. Chen and L. Wu, "A two-stage intrusion detection method based on LightGBM and autoencoder," *Mathematical Biosciences and Engineering,* p. 4765–4783, 09 February 2023.

[14] F. Khan, J. Lee and J. Han, "A hybrid neural network-based intrusion detection system using MobileNetV2 and LightGBM," *Symmetry,* p. 314, 20 February 2025.

[15] D. Wei, L. Feng and X. Zhang, "AutoEncoder and LightGBM for Credit Card Fraud Detection Problems," *Symmetry,* p. 870, 06 April 2023.

[16] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, Canberra, ACT, Australia, 2015.

[17] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue and K. Nakao, "Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation," *BADGERS '11: Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security,* pp. 29-36, 2011.

[18] P. Biondi, "Scappy," Hack.lu, Suresnes, 2005.

[19] F. Pedregosa, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* p. 2825–2830, 2011.

[20] N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research,* p. 321–357, 2011.

[21] A. Vijayvargiya, C. Prakash, R. Kumar and S. Bansal, Human Knee Abnormality Detection from Imbalanced sEMG Data, Jaipur: Biomedical Signal Processing and Control, 2021.

[22] Kali Linux, "Penetration Testing and Ethical Hacking Linux Distribution.".

[23] TensorFlow, "TensorFlow, An end-to-end open source machine learning platform.".

[24] Joblib, "Joblib: running Python functions as pipeline jobs".

[25] Nmap, "Nmap," [Online]. Available: https://nmap.org/.

[26] "Metasploit," [Online]. Available: https://www.metasploit.com/.

[27] van Hauser. [Online]. Available: https://github.com/vanhauser-thc/thc-hydra.

[28] S. Gurjar, "Medium," 4 March 2021. [Online]. Available: https://sandstorm01.medium.com/feature-scaling-in-machine-learning-understanding-normalization-and-standardization-935bd0353bd9. [Accessed 2025].

[29] H. Pelletier, "TowardsDataScience," 29 December 2023. [Online]. Available: https://towardsdatascience.com/how-to-cross-validation-with-time-series-data-9802a06272c6/. [Accessed 2025].

[30] S. Singh, R. Kumar, S. Payra and S. Kumar Singh, "Artificial Intelligence and Machine Learning in Pharmacological Research: Bridging the Gap Between Data and Drug Discovery," Cureus, Patna, 2023.

[31] Cybersecurity Ventures, "Cybercrime Report 2023," 15 February 2023. [Online]. Available: https://cybersecurityventures.com/cybercrime-report-2023/. [Accessed 16 January 2025].

[32] S. Mulani, "Regression vs Classification in Machine Learning," 5 July 2021. [Online]. Available: https://www.askpython.com/python/regression-vs-classification. [Accessed 2025].

[33] TechVidvan Team, "Reinforcement Learning Algorithms and Applications," [Online]. Available: https://techvidvan.com/tutorials/reinforcement-learning/. [Accessed 2025].

[34] Wikipedia, "Autoencoder," [Online]. Available: https://en.wikipedia.org/wiki/Autoencoder. [Accessed 2025].

[35] U. Raj, "Dropping the Knowledge Bomb: Understanding Dropout Layers in Deep Learning," 8 November 2023. [Online]. Available: https://medium.com/@utsavraj.ptn04/dropping-the-knowledge-bomb-understanding-dropout-layers-in-deep-learning-0612f517269d. [Accessed 2025].

APPENDICES

APPENDIX A: UNDERSTANDING WEIGHTS IN MACHINE LEARNING

Machine learning models use weights as fundamental parameters which establish the relative importance of input features when making predictions. The prediction process of models (linear models and neural networks) heavily depend on weights to determine how important an input is for the output result.

A.1 What are weights

Weights are numerical values that adjust the input contribution to the final prediction. For example, a widely used simple linear regression model that uses weights is:

$$y = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

Each w_i represents a weight for input feature x, and b is the bias The weight controls how much each input x will affect the output y.

A.2 Weights in Neural Networks

In neural networks, weights connect neurons between layers. For each connection:

- A weight scales the signal transmitted from one neuron to another.
- During training, the model adjusts weights using optimization algorithms (e.g., gradient descent) to minimize the prediction error.

Each neuron computes:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

Then applies an activation function f(z), such as ReLU or sigmoid, to produce its output.

A.3 Why are weights important

- **Learning Patterns**: The process of learning involves adjusting weights to better approximate the relationship between inputs and outputs.
- **Feature Importance**: Higher weights often indicate that a particular input feature is more influential.

- **Model Behavior**: The distribution and magnitude of weights directly affect how the model behaves and generalizes.

APPENDIX B: DECISION BOUNDARIES IN MACHINE LEARNING

In classification problems, a decision boundary is a critical concept that defines how a machine learning model separates data points of different classes based on their features. It is the surface or curve in the feature space that acts as a dividing line between distinct classes.

B.1 What is a decision boundary

A decision boundary is a hypersurface that partitions the feature space into regions classified as belonging to different categories. In two-dimensional data, it appears as a line (for linear models) or a curve (for nonlinear models). In higher dimensions, it becomes a plane or a more complex manifold.

For a binary classifier, the decision boundary is the set of points where the classifier is equally likely to assign a sample to either class:

$$f(x) = 0.5$$

Where f(x)f(x)f(x) is the probability estimate or decision function output.

B.2 Linear vs Non-Linear Decision Boundaries

Linear Decision Boundaries: Produced by models like logistic regression or linear support vector machines (SVMs). These create straight lines or planes that separate the classes.

Non-linear Decision Boundaries: More complex classifiers like decision trees, kernel SVMs, and neural networks can learn curved or irregular boundaries that better adapt to real-world data.

B.3 Why are decision boundaries important

- **Model Visualization**: They visualize how a model makes decisions and whether it generalizes well.
- **Important in model's generalization**: They indicate overfitting or underfitting. A too complex boundary may fit noise (overfit), while a too simple one may fail to capture patterns (underfit).

APPENDIX C: UNDERSTANDING NETWORK HOPS

In the context of computer networks, hops refer to the intermediate steps or points that a packet of data travels through as it moves from its source to its destination. Each time the packet passes through a network device (e.g., router, switch, or gateway), it is considered to have made one hop. The number of hops a packet takes is crucial in understanding network performance, latency, and efficiency.

C.1 What are Network Hops

A network hop occurs when data is transmitted from one device to another within the network. For instance:

- If a packet is sent from a computer to a router and then from the router to another router, it has made two hops.
- A hop count refers to the total number of devices the packet travels through before reaching its destination.

C.2 Types of Network Hops

- **Direct Hop**: A hop where the packet travels directly between two network devices without any intermediate routing.
- **Indirect Hop**: A hop that involves one or more intermediate routers or gateways before reaching the destination.
- **Virtual Hop**: In some cases, especially in cloud networks, hops may refer to data traveling through virtual devices or services that do not directly correspond to physical devices.

C.3 Why are network hops important

Network hops are integral to routing protocols, as they define the path a packet takes across the network. The primary role of hops is to determine the route and optimize data delivery.

- **Routing**: When a packet is transmitted over a network, routers decide the best path based on routing algorithms. Each router represents a hop in the network.
- **Latency**: More hops typically lead to higher latency, as each hop involves a small delay for routing and forwarding the packet.
- **Performance**: Fewer hops can improve performance, especially in terms of speed and efficiency, as each additional hop introduces processing time.

APPENDIX D: UNDERSTANDING ACTIVATION FUNCTIONS IN MACHINE LEARNING

Activation functions in machine learning networks specifically in neural networks determine the output values of model neurons. The functions introduce non-linear behavior to the model which enables it to detect and represent intricate patterns in data. A neural network without activation functions would operate as a linear regression model because it would fail to handle the intricate nature of most real-world problems.

D.1 What are activation functions

An activation function is a mathematical function applied to the output of a neural network neuron. It determines whether the neuron should be activated or not, based on its input. Essentially, it decides whether the information should be passed forward through the network.

D.2 Types of Activation Functions

The most common types of activation functions are:

Sigmoid Function

The sigmoid function outputs values between 0 and 1, which is ideal for binary classification problems where the output needs to be a probability.

$$f(z) = \frac{1}{1 + e^{-z}}$$

- Output values are between 0 and 1.
- Used for binary classification.
- Tends to saturate, leading to vanishing gradients for very large or small inputs.

Hyperbolic Tangent (tanh)

The tanh function outputs values between -1 and 1, providing stronger gradients than the sigmoid.

$$f(z) = \tanh(z) = \frac{2}{1 + e^{-2z}} - 1$$

- Output values are between -1 and 1.

- Similar to sigmoid but scaled.
- Still suffers from the vanishing gradient problem.

ReLU (Rectified Linear Unit)

The ReLU function has become the most popular activation function for hidden layers in deep networks. It outputs zero for negative inputs and the input itself for positive values.

$$f(z) = \max(0, z)$$

- Efficient to compute.
- Avoid vanishing gradients for positive values.
- Can cause "dead neurons" if a large portion of inputs are negative.

Leaky ReLU

A modification of ReLU, the Leaky ReLU allows a small, non-zero gradient when the input is negative.

$$f(z) = \max(az, z)$$

- Solves the dead neuron problem of ReLU.
- Allows small negative values to pass through.

D.3 Why are activation functions important

- **Non-linearity**: Activation functions allow the model to learn complex, non-linear relationships in data.
- **Control the Output**: They control the range and type of output a neuron can produce, which is critical in problems like classification, where outputs may need to be probabilities or binary values.
- **Gradient-based Learning**: Activation functions allow for effective backpropagation during training by providing gradients that are essential for updating weights.