UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Reinforcement Learning for Anomaly Detection: An Evaluation and Extension of the Adversarial Environment RL Framework

Berstärkendes Lernen zur Anomalieerkennung: Evaluation und Erweiterung des Adversarial Environment Reinforcement Learning Frameworks

Masterarbeit

verfasst am
Institut für Theoretische Informatik

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Vladislav Sehtman

ausgegeben und betreut von
Prof. Dr. Maciej Liśkiewicz

Lübeck, den 8. Mai 2025

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

_____

Vladislav Sehtman

Zusammenfassung

Mit der zunehmenden Komplexität und Vernetzung moderner IT-Systeme steigt der Bedarf an adaptiven und effizienten Intrusion Detection Systemen (IDS). Diese Arbeit untersucht das von Caminero et al. vorgestellte *Adversarial Environment Reinforcement Learning* (AE-RL) Framework, das die Interaktion zwischen einem Angreifer- und einem Verteidiger-Agenten als Reinforcement-Learning-Szenario modelliert. Aufbauend auf diesem Ansatz wurde das Framework analysiert, erweitert und in mehreren Dimensionen evaluiert.

Zu den zentralen Beiträgen zählen eine systematische Untersuchung der Modellleistung unter variierenden Trainingsdaten, architektonischen Konfigurationen sowie Multi-Agenten-Setups. Zudem wurde mit Hilfe des Boruta-Algorithmus eine umfassende Merkmalsselektion für den NSL-KDD-Datensatz durchgeführt. Die Generalisierungsfähigkeit des Modells wurde mittels Inter-Dataset-Evaluation auf den CIC-IDS-2017- und CSE-CIC-IDS-2018-Datensätzen getestet.

Die Ergebnisse bestätigen das Potenzial des AE-RL-Ansatzes insbesondere für unausgewogene Datensätze, zeigen jedoch auch Schwächen in der Generalisierung und im Sampling-Verhalten auf. Basierend auf diesen Erkenntnissen werden zukünftige Verbesserungen diskutiert, etwa durch optimierte Stichprobenstrategien, gestufte Klassifikatoren oder hybride Lernverfahren. Alle Experimente und Modelle wurden nachvollziehbar dokumentiert und sind über öffentlich zugänglichen Quellcode reproduzierbar.

Abstract

The growing complexity and scale of networked systems has amplified the need for adaptive and efficient intrusion detection systems (IDS). This thesis investigates the Adversarial Environment Reinforcement Learning (AE-RL) framework proposed by Caminero et al., which models the interaction between an attacker and a defender agent as a reinforcement learning scenario. Building on this foundation, the AE-RL framework was analysed, extended, and evaluated in multiple dimensions.

Key contributions include a systematic evaluation of the defender's performance under varying training data compositions, architectural configurations, and multi-agent scenarios. A comprehensive feature selection analysis using the Boruta algorithm was conducted to identify the most relevant input attributes for the NSL-KDD dataset. To assess the generalization capability of the model, an inter-dataset evaluation on the CIC-IDS-2017 and CSE-CIC-IDS-2018 datasets was performed.

The results confirm the AE-RL framework's potential, particularly for imbalanced datasets, but also reveal limitations in generalization and sampling consistency. Based on these insights, several improvement strategies are proposed for future work, including refined sampling policies, multi-stage classifiers, and hybrid learning architectures. All experiments and models are documented and made reproducible through publicly accessible code and training logs.

Acknowledgements

First and foremost, I would like to thank my partner Antonia Schulz, my family, and my friends — not only for their unwavering support, but also for their extraordinary patience throughout the many phases of procrastination that accompanied this thesis. Thank you for listening, encouraging, and reminding me (again and again) to finish what I started — even when I wasn't sure I would.

I am grateful to the University of Hamburg for laying the academic foundation during my Bachelor's studies, and to the University of Lübeck for helping me deepen that foundation and turn knowledge into understanding.

A special thanks goes to my supervisor, whose calm patience and thoughtful guidance helped me stay on track even when my ideas took unexpected detours. Your input was invaluable, and your trust in my process meant more than you probably know.

Thank you all — this thesis wouldn't exist without you.

Contents

1

Introduction

The rapid digitalization of modern society has led to exponential growth in the volume of data transmitted and processed across networks. Trends such as the massive deployment of Internet of Things (IoT) devices, cloud-based infrastructures, and data-driven services increasingly shape both our personal and industrial landscapes. While these developments enable smarter systems and more efficient processes, they also drastically expand the attack surface available to adversaries.

Cyberattacks on critical infrastructures — including power grids, hospitals, and research institutions — have become more frequent and sophisticated. In 2023 alone, cybercrimes of foreign origin in Germany rose by 28%, causing estimated damages of €148 billion [8]. Attacks like the Ukrainian power grid blackout (2015–2016) [3] or the IT failure at the University Hospital of Düsseldorf (2020) [19] highlight the devastating consequences of disrupted digital infrastructure. The rise of Large Language Models (LLMs) such as OpenAI's GPT-4, Google's Gemini or DeepSeek's R1 further exacerbates the threat landscape, enabling attackers to craft more convincing phishing campaigns and malware with less expertise [2].

Given this situation, Intrusion Detection Systems (IDS) have become indispensable for identifying and mitigating network-based threats. However, current IDS technologies face persistent challenges.

1.1 Intrusion Detection Systems and Their Limitations

IDS aim to detect malicious activities within host or network environments, relying either on signature-based techniques — effective against known threats but blind to novel ones — or anomaly-based techniques that model normal behaviour and flag deviations. While anomaly-based approaches are more adaptable, they tend to suffer from high false positive rates.

In addition to this tradeoff, several challenges complicate the development of robust IDS:

- Strong class imbalance in real-world data (far more benign than malicious traffic).
- A demand for fast, resource-efficient models that operate in real time.

– Poor generalization to unseen attack types or environments [21].

While many public datasets for IDS research exist, they often suffer from limited attack diversity and outdated behaviours, restricting their practical relevance. Moreover, many learning-based IDS rely on supervised methods, which require extensive labelled datasets — a resource that is difficult to obtain at scale.

1.2 Machine Learning for Intrusion Detection

To overcome these limitations, machine learning (ML) techniques have been extensively explored. Three principal paradigms are employed:

– **Supervised Learning:** Trained on labelled data, these methods can be highly effective for known attack types but generalize poorly and require costly annotation efforts.
– **Unsupervised Learning:** These models detect outliers relative to a learned baseline of "normal" behaviour. While suitable for evolving environments, they often generate many false alarms.
– **Reinforcement Learning (RL):** By learning through interaction RL adapts to changing environments. Yet its use in IDS remains limited, particularly in semi-supervised settings.

Surveys such as [1] have highlighted that supervised and unsupervised methods dominate IDS research. Semi-supervised approaches — which combine elements of both — are promising but underutilized. A noteworthy exception is the Adversarial Environment Reinforcement Learning (AE-RL) framework proposed by Caminero et al. [5], which introduces reinforcement learning to train IDS under adversarial conditions.

1.3 Motivation and Research Gap

The AE-RL framework introduces a novel training mechanism involving two RL agents: a **defender agent**, trained to classify samples, and an **attacker agent**, trained to present challenging inputs. The attacker learns to select difficult or under-represented examples, forcing the defender to improve its generalization ability. This adversarial interaction enables reinforcement-style training while still operating on labelled datasets.

Initial results on NSL-KDD and AWID showed promising performance — close to that of Support Vector Machines (SVMs) — with significantly lower inference times, suggesting AE-RL is suitable for real-time applications. However, the original work leaves several questions open:

– How does the framework behave under different training data compositions?
– How sensitive is the architecture to the size and structure of the Q-networks?
– Can the framework generalize to more recent and realistic datasets, such as CIC-IDS?
– How transparent and reproducible is the original implementation?

1.4 Objectives and Contributions of this Thesis

This thesis builds on Caminero et al.'s AE-RL approach and aims to conduct a comprehensive analysis and extension of the framework. The key contributions are:

- A **critical analysis** of the AE-RL architecture, its dynamic sampling behaviour, and limitations in the original implementation.
- An **empirical evaluation** of different training data configurations, including one-vs-one, one-vs-all, and class-balanced settings.
- An **architectural study** comparing attacker vs. defender Q-network capacity.
- A **feature selection study** using the Boruta algorithm to identify relevant features for the NSL-KDD dataset.
- An **inter-dataset evaluation** using CIC-IDS-2017 and CSE-CIC-IDS-2018, assessing generalization ability under domain shift.
- Extension of the AE-RL framework to support **multi-agent setups**, involving multiple concurrent attacker agents.

In doing so, the thesis highlights design trade-offs, implementation challenges, and practical considerations for applying reinforcement learning to intrusion detection in real-world scenarios.

1.5 Structure of the Thesis

- **Chapter 2** provides essential background on reinforcement learning, Q-learning, and Deep Q-Networks to support understanding of the proposed methods.
- **Chapter 3** introduces the AE-RL framework and summarizes related work.
- **Chapter 4** describes the utilized datasets, preprocessing steps, and label harmonization strategies.
- **Chapter 5** details the methodological contributions, training pipeline, and architectural extensions.
- **Chapter 6** presents the experimental setup, results, and analysis.
- **Chapter 7** outlines limitations and proposes directions for future research.
- **Chapter 8** summarizes the findings and main contributions of this work.

2

Preliminaries

This chapter introduces essential background concepts required for understanding the methods and algorithms used in this thesis. It is intended to provide a concise overview of the core ideas from reinforcement learning (RL), deep Q-learning, and network intrusion detection systems (NIDS). For a more detailed treatment of the theory, readers are referred to established literature.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a paradigm of machine learning in which agents learn to make sequences of decisions by interacting with an environment and receiving feedback in the form of rewards. The agent's goal is to maximize cumulative reward over time. Formally, RL problems are modelled as Markov Decision Processes (MDPs), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$:

- $\mathcal{S}$: Set of possible states
- $\mathcal{A}$: Set of available actions
- $P(s'|s,a)$: Transition probability of reaching state s' after taking action a in state s
- $R(s,a)$: Reward function
- γ: Discount factor

The agent learns a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected return. The return is typically defined as the discounted cumulative reward:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where:

- G_t: expected discounted cumulative reward at time step t
- R_{t+k+1}: reward received $k+1$ steps after time t
- $\gamma \in [0,1]$: discount factor that reduces the impact of future rewards
- k: summation index over future time steps
- t: current time step

A foundational resource for RL theory is the textbook by Sutton and Barto [17], which remains the standard reference in the field.

2.2 Q-Learning and Deep Q-Networks

Q-learning is an off-policy, value-based reinforcement learning algorithm. It learns an action-value function $Q(s, a)$ that estimates the expected return of taking action a in state s and following the optimal policy thereafter.

The update rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

In large or continuous state spaces, Q-values are approximated using neural networks—this approach is known as Deep Q-Networks (DQNs). A DQN takes the current state as input and outputs Q-values for all possible actions. The network is trained using a replay memory and mini-batch updates to improve stability.

Stabilization techniques include:

- **Experience Replay:** Stores past transitions (s, a, r, s') in a buffer to break temporal correlations.
- **Target Networks:** A separate network is used to compute target Q-values, reducing oscillations.

2.3 Adversarial Reinforcement Learning

Adversarial Reinforcement Learning (ARL) extends classical RL to settings where agents compete or interfere with each other. In such multi-agent setups, the environment becomes non-stationary from the perspective of each agent, as the actions of other agents affect the transition dynamics.

In this thesis, ARL is applied to a two-agent scenario in which one agent (the attacker) selects malicious or benign samples, and the other agent (the defender) classifies them. The attacker is rewarded when the defender misclassifies, making this a competitive game setup.

2.4 Network Intrusion Detection Systems (NIDS)

Intrusion Detection Systems are tools used to monitor and analyze network traffic for suspicious or malicious activity. NIDS can be categorized into:

- **Signature-based Detection:** Relies on known attack patterns or rules.
- **Anomaly-based Detection:** Identifies deviations from normal traffic behavior.

Anomaly-based NIDS are well suited for detecting novel threats but often suffer from high false positive rates due to the dynamic nature of network traffic.

Datasets commonly used for NIDS research include NSL-KDD, CIC-IDS-2017, and CSE-CIC-IDS-2018. These datasets consist of labelled network flow records and cover various attack types such as DoS, Probe, R2L, and U2R.

2.5 Feature Selection in Intrusion Detection

High-dimensional input features can lead to overfitting, increased training time, and reduced model interpretability. Feature selection techniques aim to identify the most informative attributes.

In this thesis, we apply the **Boruta algorithm**—a wrapper-based method that uses random forests to rank feature importance by comparing them to randomly permuted shadow features. Boruta aims to solve the *all-relevant* feature selection problem and is particularly useful for understanding which network traffic characteristics are important for detecting specific attack types.

For an in-depth discussion of the Boruta method and its impact on model performance, see Chapter 5 and Section 5.4.

2.6 Summary

This chapter provided the theoretical background necessary for understanding the reinforcement learning methods and adversarial training setup used in this thesis. Key topics such as Q-learning, deep Q-networks, adversarial RL, NIDS, and feature selection were introduced. These concepts form the basis of the methods described in Chapter 5 and evaluated in Chapter 6.

3

Adversarial Environment Reinforcement Learning Algorithm for Intrusion Detection

Since this thesis builds upon the work of Caminero et al. [5], this chapter provides a detailed overview of their research, including abstract, introduction, methods, results, and discussion.

3.1 Abstract

The authors highlight that Intrusion Detection Systems (IDS) play a vital role in modern data networks by detecting and classifying dangerous traffic. They propose a new intrusion detection algorithm that achieves high predictive accuracy. The model is based on a simple and extremely fast neural network implementing a policy function, which is trained via a novel reinforcement learning (RL) model that dynamically adapts the environment's behaviour throughout the learning process.

Traditional IDS models rely on supervised learning using predefined training data, while RL approaches typically depend on interaction with dynamic environments. The authors bridge these paradigms by introducing a simulated environment instead of a real-time one.

Their approach utilizes an intelligent simulated environment that:

- randomly extracts new samples from the training dataset,
- computes rewards based on the accuracy of the classifier's predictions, and
- adopts an adversarial objective, intentionally increasing classification difficulty.

This strategy effectively transforms the environment into a second agent in an adversarial setup, challenging the classifier. The authors demonstrate that this architecture significantly improves classification performance.

This work marks the first application of adversarial reinforcement learning for intrusion detection. The authors integrate the environment's behaviour directly into the learning process of a modified RL algorithm. Experimental evaluations show that the model is suitable for supervised learning on labelled datasets, achieving better results than conventional ML models in terms of weighted accuracy (>0.8) and F1-score (>0.79), especially excelling in the classification of under-represented labels.

3.2 Introduction

The authors emphasize the growing importance of IDS in modern data networks, particularly in response to increasing demands for security and data handling. These systems aim to efficiently analyze network traffic and detect potential threats at an early stage. IDS models are often trained using machine learning techniques to handle large, noisy, and imbalanced datasets.

A fundamental challenge in traditional IDS approaches is the assignment of intrusion labels to network features, which often requires manual annotation. While classical ML models rely on predefined datasets, RL models operate through agent-environment interaction. The authors transfer this concept to intrusion detection by modelling traffic samples as states, classifications as actions, and prediction accuracy as the reward.

Since RL models typically require dynamic environments, the authors propose using a simulated one instead. This simulated environment fulfils two main roles:

- generating samples by randomly extracting them from the dataset,
- adapting its behaviour adversarially to make classification more challenging.

Based on these mechanisms, the authors present the **Adversarial Environment Reinforcement Learning (AE-RL)** model. AE-RL combines supervised learning with an adversarial environment aimed at increasing the classifier's error rate, thereby encouraging improved differentiation between normal and anomalous patterns.

The authors place particular emphasis on the issue of imbalanced datasets, which is especially challenging for RL models. AE-RL addresses this by actively selecting harder-to-classify samples to achieve more balanced learning. This helps avoid overfitting on majority classes and improves detection of under-represented attack types.
To evaluate their approach, the authors compare AE-RL with several established ML and RL models, including Support Vector Machines (SVM), Multilayer Perceptron (MLP), Gradient Boosting Machine (GBM), and Deep Q-Networks (DQN). The experimental results show that AE-RL outperforms traditional ML approaches and static oversampling techniques such as SMOTE [6] and ADASYN [9], particularly in handling under-represented classes.

3.3 Related Work

The authors identify numerous research efforts in intrusion detection that utilize different datasets, making direct comparisons challenging due to inconsistent evaluation setups. To their knowledge, no prior work applies Deep Reinforcement Learning (DRL) with the exact premises used in this study.

A major branch of research focuses on the application of **Machine Learning (ML)** for intrusion detection. The problem can be framed either as anomaly detection or as a classification task. Several studies applied ML models to the NSL-KDD dataset, reporting, for example, a 79.9% accuracy using a Multilayer Perceptron (MLP) and a 98% F1-score with AdaBoost. Similar evaluations on the AWID dataset found that the J48 decision tree

algorithm achieved a 96% accuracy and a 94.8% F1-score. A comprehensive review of ML approaches for IDS in IoT networks is presented in [7].

Another line of research explores **Reinforcement Learning (RL)** for intrusion detection. One study describes an anomaly detector trained in a simulated environment with rewards based on correct anomaly detection. Although methodologically similar, this work uses tabular Q-learning with discrete states, which limits scalability. In contrast, modern DRL methods leverage neural networks as function approximations for continuous, high-dimensional inputs.

Multi-Agent Reinforcement Learning approaches for intrusion detection have also been studied, typically forming agent hierarchies to detect anomalies. However, these frameworks are non-adversarial and rely on discrete state representations.

Additionally, **RL for Classification** has been explored, with Actor-Critic models applied to datasets like Iris and Hepatitis. These models share some similarities with AE-RL but often extend the state space with memory cells and use complex reward schemes.

Adversarial Reinforcement Learning has seen development mainly in games or e-negotiation systems. Some studies focus on adversarial environments that mislead classifiers by subtly altering training data. While interesting, these frameworks do not target improving classification accuracy directly.

Other works model cybersecurity scenarios as zero-sum games between attackers and defenders, but using table-based Monte Carlo or Q-learning techniques, which are less scalable than DRL methods.

Finally, **handling imbalanced datasets** through oversampling techniques has been thoroughly explored. A detailed analysis of methods like Variational Generative Models (VGM), SMOTE [6], and ADASYN [9] on NSL-KDD showed improvements in classical ML classifiers. Nevertheless, AE-RL outperforms these static oversampling approaches by dynamically adapting sampling during training.

In summary, the authors position their work within the domain of RL-based intrusion detection, emphasizing the novelty of integrating an adversarial environment and dynamic resampling to address the challenges of imbalanced datasets.

3.4 Description of the Work

The authors of this work [5] propose a novel reinforcement learning model for intrusion detection that is based on adversarial environments. This section outlines the datasets used in their experiments and presents a detailed description of their approach, called *Adversarial Environment Reinforcement Learning (AE-RL)*.

Datasets Used

To evaluate their model, the authors use two well-known intrusion detection datasets: *NSL-KDD* and *AWID*. These datasets present different challenges in terms of attack frequency and distribution, allowing for comprehensive performance evaluation.

NSL-KDD Dataset

The NSL-KDD dataset is an improved version of the KDD-99 dataset, where redundant entries have been removed. It consists of *125,973 training samples and 22,544 test samples*, comprising *41 features*, 38 of which are continuous and 3 categorical. The features were normalized, and categorical features were one-hot encoded, resulting in a final dataset with *122 features*.

A key challenge of this dataset is the *significant class imbalance*: while the most frequent class represents 43.1% of the samples, the rarest class accounts for only 1.7%. This presents a difficulty for machine learning models, which tend to perform poorly on minority classes. (Note: a different class distribution was observed during our own data inspection; see Chapter 4 for details.)

For experimental purposes the dataset labels were grouped into following five main attack classes. *NORMAL*, *PROBE*, *R2L* (Remote-to-Local), *U2R* (User-to-Root), and *DoS* (Denial-of-Service). Where *NORMAL* represents normal traffic, *PROBE* includes scanning attacks, *R2L* and *U2R* represent attacks that exploit vulnerabilities in remote systems, and *DoS* includes various denial-of-service attacks. Further details are provided in Chapter 4.

AWID Dataset

The *Aegean WiFi Intrusion Dataset (AWID)* is a more recent dataset that captures both normal and malicious traffic in IEEE 802.11 wireless networks.

The authors use the *AWID-CLS-R version*, which contains four classes: *NORMAL* (no attack), *Flooding*, *Injection*, and *Impersonation*.

With *1,795,574 training samples and 575,642 test samples*, AWID is significantly larger than NSL-KDD. However, it exhibits even greater *imbalance*, with *91% of the samples being normal connections* and only 9% labelled as attacks.

Model Description

The authors introduce Adversarial Environment Reinforcement Learning (AE-RL), a novel RL-based classification model tailored for intrusion detection. The model includes a simulated environment that acts as an adversary to the classifier.

In the original work, this adversary is referred to as the "Environment Agent." For clarity, we refer to this as the "Attacker Agent," as it more accurately reflects its role in adversarial training. Correspondingly, the classifier is referred to as the "Defender Agent."

The key components of the model include:

- An **Attacker/Environment Agent** that learns to select attacks that are hard for the Defender/Classifier Agent to predict.
- A **simulated environment**, that randomly selects samples from a labelled dataset based on the attack chosen by the attacker.
- A **Defender/Classifier Agent** that attempts to correctly classify the selected attack class.

– A **reward system** where the Defender Agent receives a reward of +1 for correct classification, and no reward for incorrect predictions. The Attacker Agent is rewarded in reverse.

The model uses a traditional *Q-learning algorithm* for traffic classification. The Q-function estimates a value for each state-action pair and is updated iteratively as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_A Q(S_{t+1}, A) - Q(S_t, A_t) \right]$$

where:

– S_t is the current state,
– A_t is the selected action,
– R_{t+1} is the reward for the next step,
– α is the learning rate, and
– γ is the discount factor.

Instead of tabular Q-learning, the authors use *neural networks* as function approximations, making the model suitable for high-dimensional input data.

Adversarial Learning Strategy

A central element of AE-RL is the *adversarial environment*, specifically the embedded Attacker, which intentionally selects challenging samples to improve the classifier. This is achieved through:

– The Attacker's ability to observe that frequently selecting rare or difficult samples *increases the error rate* of the defender.
– An *adaptive sampling strategy* that is optimized during training.
– *Weight adjustments* based on the classifier's past prediction errors.

This results in a *more balanced learning strategy*, particularly for rare attack types that traditional models struggle to detect.

Dynamic Data Adaptation

While classical oversampling techniques like *SMOTE* [6] or *ADASYN* [9] create synthetic samples statically, AE-RL "generates" samples *dynamically during training*. This allows the model to retain the original data structure and adapt to *changing threat landscapes*.

The authors compare AE-RL against various established algorithms, including:

– **Machine Learning:** Support Vector Machines (SVM), Random Forest, AdaBoost and Multilayer Perceptron (MLP).
– **Deep Learning:** Convolutional Neural Networks (CNN).
– **Reinforcement Learning:** Double Deep Q-Networks (DDQN), Dueling DDQN, Asynchronous Advantage Actor-Critic (A3C).

Their experiments show that AE-RL achieves *higher classification accuracy* on *imbalanced* datasets than conventional approaches.

3.5 Results

The authors compare the performance of their proposed *AE-RL* algorithm with various Machine Learning (ML), Deep Learning (DL), and Deep Reinforcement Learning (DRL) models on the *NSL-KDD* and *AWID* datasets. The models compared include:

- **Machine Learning:** Logistic Regression, Support Vector Machine (SVM) with linear and RBF kernels, Random Forest (RF), Gradient Boosting Machine (GBM), AdaBoost.
- **Deep Learning:** Multilayer Perceptron (MLP) and Convolutional Neural Networks (CNN).
- **Reinforcement Learning:** Double Deep Q-Network (DDQN), Dueling DDQN, Asynchronous Advantage Actor-Critic (A3C).

Performance Evaluation

All experiments are conducted on the NSL-KDD and AWID datasets. The following metrics are used for performance evaluation:

- **Accuracy**
- **F1-Score**
- **Precision**
- **Recall**

Due to the strong class imbalance present in the datasets, the *weighted F1-Score* is considered the most important metric, as it better handles the minority classes than pure accuracy.

Classification Performance Comparison

The aggregated results for the *NSL-KDD* dataset show that *SVM with RBF kernel* achieves the best F1-scores, followed closely by *AE-RL*, which, however, requires significantly less computation time. *AE-RL offers comparable F1 performance while achieving much faster inference times*, making it highly suitable for practical intrusion detection system deployments.

The One-vs-Rest analysis shows that AE-RL is particularly strong in detecting rare classes. Although the detection rate is high across all classes, the number of false negatives in rare classes is significantly reduced, which is critical in IDS scenarios.

Comparison with Oversampling Methods

The authors also compare the performance of AE-RL with traditional ML methods enhanced by oversampling techniques such as *VGM, SMOTE, ADASYN, and Easy Ensemble*. The results show that AE-RL achieves *higher F1-Scores* than even the best-performing oversampling strategies, particularly for under-represented attack types.

Computational Time

A key factor for the practical applicability of an IDS algorithm is the *computation time for training and inference*.

The results show that AE-RL achieves *comparable or better prediction accuracy than SVM-RBF* while requiring *significantly less computation time*. While SVM-RBF involves complex decision boundary calculations, AE-RL relies on a simple neural network, enabling much faster inference times. This characteristic is particularly critical for real-time cybersecurity applications.

Dynamic Environment Adaptation

The authors also examine the optimization of the adversarial environment's *exploration* behaviour. The best F1-Score is achieved when the *environment's epsilon-greedy threshold is maintained around 0.8 during training*, indicating that retaining some degree of randomness in environment control improves model robustness.

AWID Dataset: Model Comparison

The authors compare the *confusion matrix of the AE-RL model* against two established algorithms for the *AWID* dataset: a *MLP model* with a similar architecture to AE-RL and the *J48 decision tree* algorithm.

The results show that *AE-RL has the lowest number of false negatives for the Impersonation and Flooding attack types*. While J48 achieves high overall accuracy, it suffers from a *93% error rate for Impersonation attacks*, which poses a significant security risk in IDS systems where missed attacks can have severe consequences.

Summary of the Results

The key findings from the experiments are:

- *AE-RL achieves classification performance comparable to the best existing models while requiring significantly less computational time.*
- *AE-RL excels in handling imbalanced datasets by improving the classification of rare attack types.*
- *The adaptability of the adversarial environment allows AE-RL to dynamically adjust to evolving threats.*
- *The model is particularly suitable for real-time IDS deployments, providing fast and precise predictions.*

3.6 Conclusion

Intrusion Detection Systems (IDS) are an essential component of modern data networks, especially as security threats continue to evolve and are increasingly difficult to attribute to specific traffic patterns. The authors argue that traditional Machine Learning models

face several challenges, such as dealing with noisy, imbalanced, and dynamically changing data, which limits their effectiveness. Therefore, it is necessary to explore new models capable of addressing these limitations.

One promising approach is the application of Reinforcement Learning (RL), which has already proven successful in other domains like robotics, finance, and gaming. The authors demonstrate that RL principles can also be effectively transferred to IDS problems. Their proposed model, *Adversarial Environment Reinforcement Learning (AE-RL)* [5], combines supervised learning with an adversarial RL framework. The core idea is to create a simulated environment that:

1. interacts with a dataset of pre-recorded network features and corresponding intrusion labels,
2. strategically selects samples to maximize the classifier's learning performance.

This work represents the *first application of adversarial reinforcement learning for intrusion detection*. The authors present a novel technique that integrates the environment's behaviour directly into the learning process of a modified RL algorithm. They further propose an innovative *oversampling strategy* for under-represented categories, which has proven beneficial for training RL-based models.

Key Contributions

The key contributions of this work [5] can be summarized as follows:

- Development of a novel architecture combining supervised learning with adversarial reinforcement learning, tailored for complex network environments.
- Demonstration that AE-RL achieves classification performance comparable to non-linear models like SVM-RBF, while requiring *significantly lower inference times*.
- Introduction of a model trainable with differentiable loss functions on high performance platforms (e.g., TensorFlow), despite using a non-differentiable underlying reward function and enhancing model *flexibility* and *adaptability*.
- Comprehensive comparison of AE-RL with multiple established ML methods for intrusion detection, with a focus on *classification performance, computational times, and model scalability*.
- Provision of a model specifically optimized for *imbalanced datasets* by dynamically prioritizing hard-to-classify samples during training.

Future Research Directions

The authors suggest several promising directions for future research. One interesting extension would be applying the intelligent environment concept to *other classification problems*. Furthermore, the environment could be refined to act as a *prioritized experience replay* mechanism, with an additional agent optimizing the sampling strategy using RL principles.

In conclusion, this work lays the foundation for the advancement of adversarial reinforcement learning methods in the field of network security and demonstrates their potential for significantly improving intrusion detection systems.

3.7 Critical acclaim

The work presents an innovative approach to applying reinforcement learning to intrusion detection, characterized by several unique features. The combination of supervised learning with adversarial environments is a promising approach to improve the classification performance of IDS systems. The authors demonstrate that their proposed AE-RL model achieves high accuracy in detecting attacks while requiring significantly less computation time compared to similar models.

Critical Analysis of the Episode Definition and Sampling Strategy

In their original work [5], the authors explicitly state:

"We consider an episode as a training round throughout the entire dataset, in this case another usual denomination for episode is epoch."

Upon detailed code-level examination, however, this statement does not match their practical implementation:

1. Initialization and Sampling per Episode

Each episode initializes at a randomly chosen index within the training dataset:

$$self.index = np.random.randint(0, self.df.shape[0]-1)$$

With the authors' parameters (iterations_episode = 100, batch_size = 1), each episode uses exactly 100 samples, representing approximately 0.08% of the full NSL-KDD training set (≈125,000 samples). This contradicts their definition of an episode covering the entire dataset.

2. Dataset Boundary Handling

When reaching the dataset's end, the sampling wraps around, but the episode length remains constant (100 samples), clearly insufficient for complete dataset coverage within a single episode.

3. Learning Procedure with Experience Replay

The agents' actual training utilizes Experience Replay:

$$(states, actions, rewards, next\_states, done) = self.memory.sample\_minibatch(self.minibatch\_size)$$

This method inherently introduces variability. The Replay Memory of each Agent utilized in the approach has a limited capacity which is *set to 1000 experiences*. After the first episode, the memory is filled with 100 samples and reaches the predefined threshold of *minibatch_size = 100*. Every consecutive iteration of each episode triggers the random sampling of 100 samples from the replay memory, on which the training is performed. After the tenth episode, the memory is filled with 1000 samples and hits the capacity limit. From this point, the replay memory is updated by removing the oldest samples and adding the new ones in a ring buffer fashion.

With 100 samples generated per episode and 100 episodes of training, a maximum of 10,000 samples from the training set can be randomly chosen. However, due to the nature of the sampling process, the same samples may be selected multiple times, while

others may not be selected at all. There is no guarantee that all samples are selected at least once during training.

Consequently, the effective training set at any point during the learning phase is considerably smaller. Thus, each episode is effectively a random partial traversal of the dataset, rather than a complete epoch.

Proposed Enhancement to Sampling Methodology

Despite the aforementioned inaccuracies the authors' strategy of random dynamically sampling training data provides essential flexibility especially beneficial for addressing highly imbalanced anomaly detection datasets. However, the randomness in sample selection complicates the reproducibility of their results. During the experiments, variations of approximately 1–12% in the model's performance (F1-Score, Accuracy, Precision, Recall) were consistently observed across different training runs, attributable to this random sampling strategy.

To mitigate such variability while preserving the advantages of dynamic sampling we propose introducing a more controlled sampling mechanism. This mechanism should *maintain randomness* to ensure adaptability to unbalanced data but also *guarantee* that *every sample* from the dataset is *selected at least once during training*. Such an approach aims to retain the method's strengths—effective handling of imbalanced datasets—while improving reproducibility and ensuring comprehensive utilization of available data.

The full utilization of the dataset should make this approach more robust against overfitting and underfitting, as the model will be trained on a more diverse set of samples. Additionally, the resulting model will likely generalize better to unseen data and can be more effectively compared to other paradigms, as they are trained on the full dataset. This approach could further enhance the model's performance and is worth exploring in future research.

4

Used Datasets

In this chapter, we provide an overview of the datasets, KDD-1999, NSL-KDD, CIC-IDS-2017 and CSE-CIC-IDS-2018 [15] used in this thesis, including their origins, characteristics, and preprocessing steps. First, we discuss the NSL-KDD dataset, which serves as the primary dataset for our experiments, as it was the basis for the AE-RL framework proposed by Caminero et al. [5]. The NSL-KDD dataset is a widely used benchmark for evaluating intrusion detection systems, and it has been extensively studied in the literature especially in the past two decades. Although it has some limitations and is considered outdated, it remains a valuable resource for understanding the challenges of intrusion detection and for developing and testing new algorithms that can be better compared to existing ones. Therefore we will first provide a detailed overview of the NSL-KDD dataset, including its features and characteristics.

We will not reuse the AWID dataset, even though it is also used in the original AE-RL paper [5], instead we will use the CIC-IDS datasets, which are more recent and comprehensive datasets for evaluating intrusion detection systems. The Canadian Institute for Cybersecurity (CIC) released several datasets that are widely used in the research community [11]. The CIC-IDS [15] datasets are designed to reflect real-world network traffic and include a variety of attack scenarios, making them ideal for evaluating intrusion detection systems in realistic environments. In addition, we will make an inter-set model evaluation, which means we will train the model on one dataset and evaluate it on another dataset. The CIC-IDS-2017 dataset will be used for training and the CSE-CIC-IDS-2018 dataset for evaluation, and vice versa in order to test the generalization capabilities of the model like it was done in [21].

As the authors of the AE-RL paper did not provide any information about the feature selection process, we will also provide a detailed overview of the feature selection process we used for the NSL-KDD dataset. Specifically, we employed the Boruta feature selection algorithm, a wrapper method based on the random forest classifier. We have chosen this method because it is a well-established and widely used feature selection technique that has been shown to be effective in various domains, including intrusion detection.

We will also provide a detailed overview of the preprocessing steps we applied to the datasets, including data cleaning, normalization, and feature engineering. The NSL-KDD dataset [18] used in this thesis is a revised version of the KDD'99 dataset [16]. Thus, examining the KDD'99 dataset helps to understand its features and the improvements

introduced in the NSL-KDD dataset.

4.1 Origin of the NSL-KDD Dataset: KDD-1999

The DARPA Intrusion Detection Evaluation Program of 1998, organized by MIT Lincoln Laboratory, aimed to evaluate research efforts in intrusion detection [16].

To this end, a standardized dataset was created containing various simulated attacks within a military network environment. The KDD'99 Intrusion Detection Contest used a revised version of this dataset. Lincoln Labs collected nine weeks of raw TCP dump data from a simulated US Air Force LAN, which operated with realistic network traffic but was intentionally subjected to attacks.

The training data consisted of four gigabytes of compressed TCP dump data over seven weeks of network activity, from which five million connection records were generated. The two weeks of test data produced an additional two million connection records.

A *connection* is defined as a sequence of TCP packets between a source and destination IP address under a specific protocol. Each connection was labelled either as "normal" or as an attack with a specific attack type. The attack types were categorized into four main groups:

1. DoS (Denial-of-Service): e.g., SYN Flood
2. Probing: reconnaissance and scanning activities, e.g., Port Scanning
3. R2L (Remote-to-Local): unauthorized remote access, e.g., Password Guessing
4. U2R (User-to-Root): unauthorized root access, e.g., Buffer Overflow

An important characteristic is that the *test data* had a different distribution than the *training data* and contained additional attacks not present during training, thereby simulating a more realistic scenario. Overall, the dataset included 24 attacks in the training set and 14 additional attacks in the test set.

Stolfo et al. [16] developed higher-level features to distinguish normal connections from attacks. These features can be categorized as follows:

- Time-based Features:
  - "Same Host": analyses all connections over the last two seconds to the same destination host and computes statistical properties regarding protocol behaviour, services, etc.
  - "Same Service": analyses all connections over the last two seconds using the same service.
- Host-based Features: Some attacks, such as port scans, occur over extended periods (e.g., one scan per minute). Therefore, connections were grouped by destination host, and features were computed over the last 100 connections rather than using a time window.
- Content-based Features: R2L and U2R attacks are usually hidden within packet data fields and are often observable within a single connection. To detect suspicious behaviour, features such as the number of failed login attempts were introduced, which explicitly incorporate expert knowledge in the field of intrusion detection.

The following tables (4.1, 4.2, 4.3) list all the features of the dataset in detail [16].

Table 4.1: Basic features of individual TCP connections.

Feature Name	Description	Type
duration	Duration of the connection (in seconds)	continuous
protocol_type	Type of protocol (e.g. TCP, UDP)	discrete
service	Target network service (e.g. HTTP, Telnet)	discrete
src_bytes	Number of bytes from source to destination	continuous
dst_bytes	Number of bytes from destination to source	continuous
flag	Normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; otherwise 0	discrete
wrong_fragment	Number of "wrong" fragments	continuous
urgent	Number of urgent packets	continuous

Table 4.2: Content-Based features within a TCP connection, based on domain knowledge.

Feature Name	Description	Type
hot	number of "hot indicators"	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; otherwise 0	discrete
num_compromised	number of "compromised" conditions	continuous
root_shell	1 if root shell is obtained; otherwise 0	discrete
su_attempted	1 if "su root" command attempted; otherwise 0	discrete
num_root	number of "root" accesses	continuous
num_file_creations	number file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in a ftp session	continuous
is_hot_login	1 if the login belongs to the "hot" list; otherwise 0	discrete
is_guest_login	1 if the login is a "guest"login; otherwise 0	discrete

A summary about the original KDD dataset is available at `https://kdd.ics.uci.edu/databases/kddcup99/task.html`.

4.2 The NSL-KDD Dataset

The NSL-KDD dataset is a widely used dataset for Intrusion Detection Systems and represents an improved version of the KDD-99 dataset described in the previous section 4.1. NSL-KDD addresses the issue of redundant entries in the original dataset [18].

It contains 125,973 training samples and 22,544 test samples, each with a total of *41 features*: 38 continuous and 3 categorical features. These features have been further processed. Continuous features were scaled to the range [0–1] using min-max normaliza-

Table 4.3: Traffic Features Based on a Two-Second Time Window

Feature Name	Description	Type
count	number of connections to the same host as the current connection in the past 2 seconds *Note: The following features refer to these same host connections.*	continuous
serror_rate	% of connections that have "SYN" errors	continuous
rerror_rate	% of connections that have "REJ" errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past 2 seconds *Note: the following features refer to these same-service connections.*	continuous
srv_serror_rate	% of connections that have "SYN" errors	continuous
srv_rerror_rate	% of connections that have "REJ" errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

tion, while categorical features were transformed using one-hot encoding, following the original work [5].

The final training dataset includes 38 continuous features and 84 binary features (0, 1), which result from the one-hot encoding of the categorical features: *protocol_type*, *flag*, and *service*. Specifically, *protocol_type* has 3 possible values, *flag* has 11 possible values, and *service* has 70 possible values. Each unique value in these features is represented as a separate binary column after one-hot encoding, resulting in a total of $3 + 11 + 70 = 84$ binary columns. Details about the possible values of these features are provided in Appendix A.3. A complete overview of attack types, specific attacks, and their frequencies is provided in Appendix A.

Min-max normalization is a common preprocessing technique used to scale continuous features to a fixed range, typically $[0, 1]$. This ensures that all features contribute equally to the model's learning process, regardless of their original scale. Each value x is transformed using the formula:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

where $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of the feature vector, respectively, and x' is the normalized value. This transformation preserves the relationships between values while ensuring that all features lie within the same range, improving the stability and performance of machine learning models.

One-hot encoding is a commonly used method to convert categorical data into a numerical format suitable for machine learning models. For example, consider the categorical feature *protocol_type*, which has three possible values: *tcp*, *udp*, and *icmp*. Using one-hot encoding, this feature is replaced by three binary columns: *tcp*, *udp*, and *icmp*.

If a sample originally had the value *tcp*, it would be represented as [1, 0, 0], where the first column (*tcp*) is set to 1, and the others are set to 0. Similarly, a sample with the value *udp* would be encoded as [0, 1, 0]. This transformation ensures that the categorical data is represented numerically without introducing any ordinal relationships between the categories.

After the preprocessing the final dataset has 122 features.

The dataset is highly imbalanced, which is typical for anomaly detection datasets, where normal traffic significantly outweighs anomalous instances. In the *entire dataset* (training + test), the most frequent class *normal* accounts for 51.9%, while the rarest class *U2R* occurs in only 0.17% of samples, as shown in Table 4.4.

Table 4.4: Attack types in the *entire* NSL-KDD dataset.

Attack Type	Count (Total = 148,517)	Rounded share in %
normal	77.054	51,9 %
DoS	53.385	35,9 %
Probe	14.077	9,48 %
R2L	3.749	2,52 %
U2R	252	0,17 %

The table shows the distribution of different attack types across the entire dataset. The training set contains 125,973 samples and the test set 22,544 samples.

In the *training set*, the most frequent class *normal* accounts for 53.5%, while *U2R* is the rarest at 0.04%, as shown in Table [4.5].

Table 4.5: Overview of attack types and their distribution in the *training dataset*.

Attack Type	Individual Attacks	Count (Total = 125,973)	Share %
Normal	normal	67.343	53,46 %
DoS	back, land, neptune, pod, smurf, teardrop	43.727	36,46 %
Probe	ipsweep, nmap, portsweep, satan	11.656	9,25 %
R2L	ftp_write, guess_passwd, imap, multihop, phf, spy, warezclient, warezmaster	995	0,79 %
U2R	buffer_overflow, loadmodule, perl, rootkit	52	0,04 %

In the *test set*, *normal* accounts for 43.1% and *U2R* for 0.89%, as shown in Table [4.6].

In contrast to the original work [5] (section 3.1.1), our analysis revealed slightly different frequencies for attack types. According to the original paper, the most common class (unspecified) accounts for 43.1% of the data, which matches the *test set* value in our analysis. However, their rarest class is reported as 1.7%, which does not align with any of our computed values.

Each training sample is assigned to one of 23 attack labels (one "normal" and 22 attack labels). The test set contains the same number of features (41 or 122 after transformation), but the labels span 38 different attack labels (one "normal" and 37 attack labels).

This indicates that the test data contains anomalies that were not present in the train-

Table 4.6: Overview of attack types and their distribution in the *test dataset*.

Attack Type	Individual Attacks	Count (Total = 22,544)	Share %
Normal	normal	9.711	43,08 %
DoS	back, land, neptune, pod, smurf, teardrop, mailbomb, apache2, processtable, udpstorm	7.458	33,08 %
Probe	ipsweep, nmap, portsweep, satan, mscan, saint	2.421	10,74 %
R2L	ftp_write, guess_passwd, imap, multihop, phf, warezmaster, sendmail, named, snmpgetattack, snmpguess, xlock, xsnoop, worm	2.754	12,22 %
U2R	buffer_overflow, loadmodule, perl, rootkit, httptunnel, ps, sqlattack, xterm	200	0,89%

ing set. Of the 23 training and 38 test labels, 21 are shared between them, while 2 appear only in training and 17 only in testing (for detailed information about the labels see Appendix A.3). According to [5], up to 16.6% of the test data contain labels not present during training, posing an additional challenge for learning models as the model must generalize to unseen attacks.

For better interpretability, the 23 labels were consolidated into meaningful classes. As described in [18] and [16], both training and test labels can be mapped to five classes: *normal, DoS, Probe, R2L,* and *U2R*. All classes except *normal* indicate an attack.

These five classes were used as final labels in evaluation as they help to characterize attacks, maintain the dataset's inherent imbalance (which is essential in anomaly detection), and ensure that the number of samples per class is large enough for meaningful results [5].

The concrete distributions of attack classes and individual attacks in the NSL-KDD training dataset are illustrated in Figures 4.7 and 4.8.

4.3 Analysis of R2L and U2R Attacks in the NSL-KDD Dataset

When calculating the proportions of attack classes (normal, DoS, Probe, R2L, U2R) for both training and test sets, we observed that not only do the types *R2L* and *U2R* have disproportionately low representation, but also the individual attacks within them.

While it is useful to have different attacks in training and test sets (see Appendix A.3) to evaluate how models generalize to unknown attacks, the proportions of shared attack types between training and testing should be approximately consistent.

Training on highly unbalanced data is not useful if the model lacks enough samples to learn the underlying patterns of rare attacks. This partly explains why the AE-RL model in the original work [5] performed poorly on *Remote-to-Local (R2L)* and *User-to-Root (U2R)* attack types.

All statistics related to attack types and attacks can be found in Appendix A.

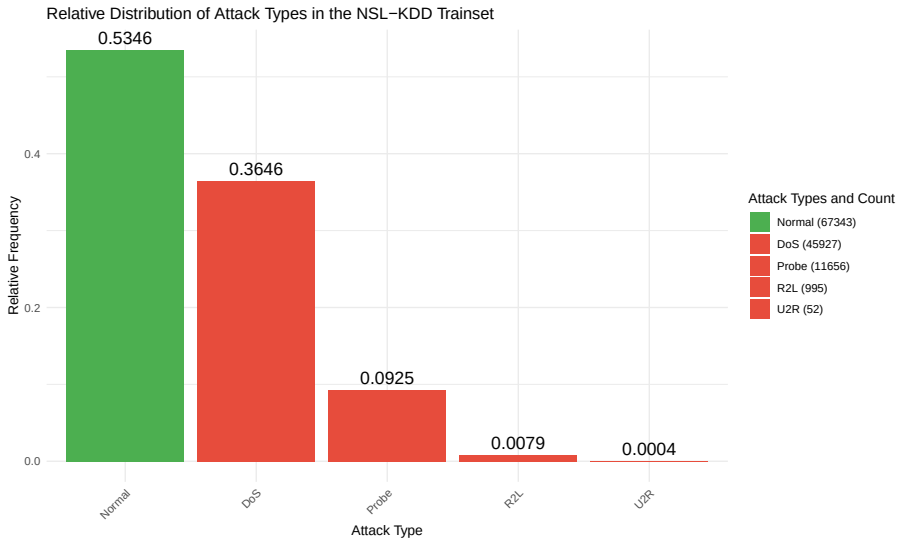

Figure 4.7: Distribution of attack classes in the NSL-KDD training dataset.

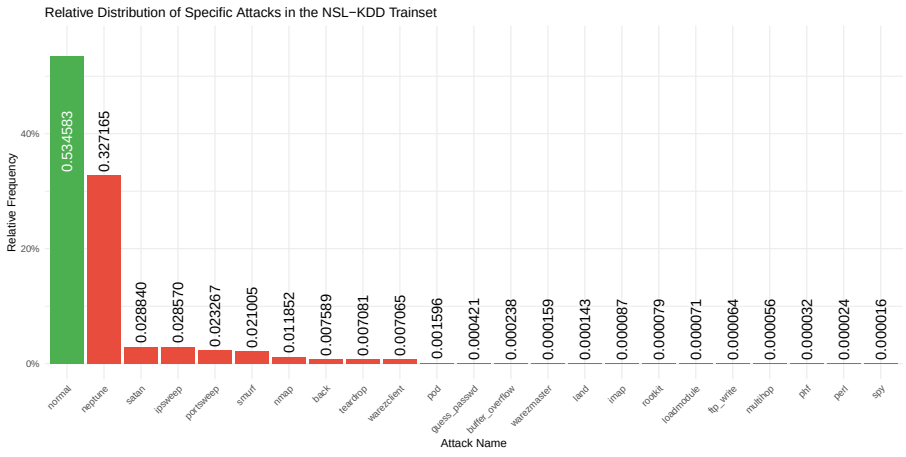

Figure 4.8: Distribution of individual attacks in the NSL-KDD training dataset.

R2L Attacks

In the training dataset, there are a total of *995 Remote-to-Local (R2L) instances*.

Two attacks, *spy* with only 2 instances and *warezclient* with 890 instances, occur exclusively in the training dataset.

In contrast, the test dataset contains *2754 R2L instances*, several of these attacks are present *only in the test set*.

These include:

- *sendmail* 14 instances,
- *named* 17 instances,
- *snmpgetattack* 178 instances,
- *snmpguess* 331 instances,
- *xlock*: 9 instances
- *xsnoop* 4 instances
- *worm* 2 instances

Since these attacks are not present in the training data, the model is exposed to entirely new attack patterns during testing, providing a meaningful evaluation of its ability to generalize to unknown threats.

In addition to attacks unique to either the training or test datasets, there are several that appear in *both*, but with significantly different instance counts:

- *ftp_write*: 3 test instances / 8 training instances
- *guess_passwd*: 1231 test instances / 53 training instances
- *imap*: 1 test instance / 11 training instances
- *multihop*: 18 test instances / 7 training instances
- *phf* : 2 test instances / 4 training instances
- *warezmaster*: 944 test instances / 20 training instances

Notably, some attacks appear in the test set at rates many times higher than in the training set. For example, *guess_passwd* appears *1231 times* in the test set but only *53 times* in training — approximately a *23-fold increase*. Even more extreme is *warezmaster*, which has *944 test instances* but only *20 training instances* — a *47-fold increase*.

U2R Attacks

Similar to the R2L category, the *User-to-Root (U2R)* category also displays a striking discrepancy in distribution between training and test datasets.

The number of *U2R training instances* is very low in general, making it difficult for a model to effectively learn and detect these attacks. There are only *52 U2R instances* in the training set, compared to *200 instances* in the test set.

Some U2R attacks are exclusive to the test dataset, including:

- *httptunnel*: 133 instances
- *ps*: 15 instances
- *sqlattack*: 2 instances
- *xterm*: 13 instances

These attacks were not encountered during training, forcing the model to rely entirely on generalized learning to detect them — again serving as a direct evaluation of the classifier's *generalization ability*.

There are also U2R attacks that appear in both training and test sets with the following distributions:

- *buffer_overflow*: 20 test instances / 30 training instances
- *loadmodule*: 2 test instances / 9 training instances
- *perl*: 2 test instances / 3 training instances
- *rootkit*: 13 test instances / 10 training instances

For *rootkit*, the number of test instances is slightly higher, although not as dramatically different as seen in some R2L attacks.

Evaluation of the Current Data Distribution and Creation of a Balanced Dataset

The current distribution of R2L and U2R attacks raises concerns, particularly regarding the highly imbalanced occurrence of R2L attacks. While placing certain attacks exclusively in the test set can simulate realistic threat conditions, extreme discrepancies in the number of training and test instances for individual attacks can be problematic. We first analyze the test-to-training ratios for each attack type and then propose a solution to improve R2L and U2R training coverage, resulting in a new *balanced dataset*.

If we sum all R2L attacks that appear exclusively in the test set, we get 555 instances, accounting for about 20% of the total R2L test data (see Table 4.10). For U2R attacks, 163 instances appear only in the test set, which corresponds to approximately 81.5% of all U2R test instances (see Table 4.12).

Table 4.9 compares the test/training instance ratios for all attack categories.

Table 4.9: Ratios of attack types in the *test dataset*/*training dataset*.

Attack Type	*test instances*/*training instances*	Ratio
Normal	9711/67343	0,1442
DoS	7458/43727	0,1706
Probe	2421/11656	0,2077
R2L	2754/995	2,7678
U2R	200/52	3,8462

Clearly, the R2L and U2R categories have significantly higher test-to-training ratios (≈2.77 and ≈3.85, respectively) compared to the other categories (≈0.14 to ≈0.21). This indicates a substantial under-representation of R2L and U2R attacks in the training set.

To address this, we propose transferring selected R2L and U2R instances from the test set to the training set aiming to improve training coverage while maintaining the anomaly nature of the dataset.

To reach a target ratio of ≈ 0.4 for R2L, 1675 instances need to be transferred from test to training. This was calculated using: $\frac{Test_{Amount\ instances}-x}{Training_{Amount\ instances}+x} \approx 0,4$

where x represents the number of instances transferred from the test to the training set. Solving for x gives us $x \approx 1675$.

When selecting which R2L attacks to move, it is useful to examine individual ratios (Table 4.10). Notably, *guess_passwd* and *warezmaster* occur in both sets with a high number of test instances. Thus, 950 *guess_passwd* and 725 *warezmaster* instances were transferred from test to training. Importantly, this redistribution does not introduce new attack types into the training set or remove any from the test set, ensuring the integrity of the evaluation process. The updated distribution of R2L attacks after redistribution is shown in Table 4.11.

Table 4.10: Distribution of R2L attacks between training and test datasets.

Attack	Ratio in test dataset/training dataset	Type
spy	0/2	only in training
warezclient	0/890	only in training
ftp_write	3/8	in both datasets
guess_passwd	*1.231/53*	in both datasets
imap	1/11	in both datasets
multihop	18/7	in both datasets
phf	2/4	in both datasets
warezmaster	*944/20*	in both datasets
sendmail	14/0	only in test
named	17/0	only in test
snmpgetattack	178/0	only in test
snmpguess	331/0	only in test
xlock	9/0	only in test
xsnoop	4/0	only in test
worm	2/0	only in test

For U2R, direct redistribution of *shared* attacks is insufficient due to the low combined count (89 instances for *buffer_overflow*, *loadmodule*, *perl*, and *rootkit* 4.12).

Therefore, 118 of the 133 *httptunnel* instances (which previously existed only in the test set) were transferred to the training set. Additionally, all 30 *buffer_overflow* instances of the training set were reassigned to the test set, and 10 of 13 *rootkit* instances were transferred to training. That way we keep the number of attacks that are only in the test set (4) and do not reduce the number of attacks that are only in the training set (2). The resulting split gives us 102 U2R test instances and 150 in training, for a ratio of ≈ 0.68 (Table 4.13).

Table 4.14 shows the updated distribution across all categories. The overall training/test ratio changes from 125,973/22,544 ≈ 0.1790 to 127,746/20,771 ≈ 0.1626 — a minor shift of 0.0164.

That dataset is still highly imbalanced, but the ratio of R2L and U2R attacks is now more balanced therefore we will refer to it as the *balanced dataset*.

Table 4.11: *New* distribution of R2L attacks between training and test datasets.

Attack	Ratio in Test Dataset/Training Dataset	Type
spy	0/2	only in training
warezclient	0/890	only in training
ftp_write	3/8	in both datasets
guess_passwd	*281/1003*	in both datasets
imap	1/11	in both datasets
multihop	18/7	in both datasets
phf	2/4	in both datasets
warezmaster	*219/745*	in both datasets
sendmail	14/0	only in test
named	17/0	only in test
snmpgetattack	178/0	only in test
snmpguess	331/0	only in test
xlock	9/0	only in test
xsnoop	4/0	only in test
worm	2/0	only in test

Table 4.12: Distribution of U2r attacks between training and test datasets.

Attack	Ratio in test dataset/training dataset	Type
buffer_overflow	20/30	in both datasets
loadmodule	2/9	in both datasets
perl	2/3	in both datasets
rootkit	13/10	in both datasets
httptunnel	133/0	only in test
ps	15/0	only in test
sqlattack	2/0	only in test
xterm	13/0	only in test

Table 4.13: *New* distribution of U2r attacks between training and test datasets.

Attack	Ratio in test dataset/training dataset	Type
httptunnel	*15/118*	in both datasets
loadmodule	2/9	in both datasets
perl	2/3	in both datasets
rootkit	*3/20*	in both datasets
buffer_overflow	*50/0*	only in test
ps	15/0	only in test
sqlattack	2/0	only in test
xterm	13/0	only in test

Table 4.14: *New* ratios of attack types in the *test dataset/training dataset*.

Attack Type	*test dataset/training dataset*	Ratio
Normal	9711/67343	0,1442
DoS	7458/45927	0,1624
Probe	2421/11656	0,2077
R2L	*1079/2670*	0,4041
U2R	*102/150*	0,68

Final Remarks

To the best of my knowledge, there is no universally accepted estimate for the ratio of malicious to normal traffic in real-world networks. This ratio varies significantly depending on factors such as network architecture, industry, geographic location, and implemented security practices.

The NSL-KDD dataset provides a structured categorization of attacks into DoS, Probe, R2L, and U2R, which is highly valuable for research and benchmarking. However, it is important to recognize that this dataset reflects a simulated environment and serves as an approximation of real-world systems. While it offers a useful foundation for developing and testing intrusion detection models, its attack distributions and scenarios may not fully capture the complexity and variability of operational networks.

In practice, the frequency and nature of attacks are dynamic and context-dependent. Detection strategies must therefore remain adaptable to evolving threats. Preparing anomaly detection models to handle a wide variety of attack patterns is crucial for ensuring robust protection. Although this complicates benchmarking, it enhances the model's applicability in real-world scenarios.

The extreme under-representation of R2L and U2R attacks in the training set relative to the test set poses a significant challenge. Increasing the number of training samples for these categories would provide the model with greater variance and improve its ability to generalize, particularly for reinforcement learning approaches.

With the current imbalance, models may struggle to correctly classify rare attacks simply because they are under-represented during training. Redistributing certain instances — especially from shared attacks that are overrepresented in the test set — is a practical approach to address this issue.

Ultimately, the appropriateness of such redistribution depends on the experiment's objectives. If the goal is to expose the model to a diverse range of attack patterns, redistribution is advisable. Conversely, if the aim is to evaluate the model's ability to generalize to entirely novel attacks, the current split should be preserved.

An alternative strategy could involve clustering attacks based on their features rather than relying solely on specific attack names. This approach may enable the model to learn broader behavioural patterns, improving its ability to detect novel attacks more effectively.

4.4 Preprocessing of the NSL-KDD Dataset

The preprocessing of the NSL-KDD dataset was conducted following the methodology described in the original work [5] and the corresponding GitHub repository [4]. The steps are outlined below:

1. Removal of the 'difficulty' Column

The *'difficulty'* column was completely removed, as it was deemed irrelevant for the training process.

2. One-Hot Encoding of Categorical Features

The categorical features *protocol_type*, *service*, and *flag* were one-hot encoded to transform them into numerical representations suitable for machine learning models. One-hot encoding creates a binary column for each unique value in the categorical feature. See section 4.2 for an example of one-hot encoding.

3. Handling Faulty Values

The feature *su_attempted* contained unexpected values that were not documented in the original NSL-KDD dataset. Following the approach in [5], these values were assumed to be errors and were set to 0.

4. Labels and Final Data Structure

The labels column was removed and replaced by one-hot-encoded labels. The resulting formatted dataset contains 162 columns: 122 features and 40 one-hot-encoded label columns. The label columns were excluded from the training process but retained for evaluation purposes.

5. Normalization of Continuous Features

Continuous features were normalized to the range [0, 1] using min-max scaling (4.2). Each value x was transformed using the formula:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

where $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of the feature, respectively. For the feature *num_outbound_cmds*, which had constant values of 0, the entire column was set to 0 to maintain consistency.

4.5 Final Feature Set: NSL-KDD

After applying the preprocessing steps described in Section 4.4, the NSL-KDD dataset was transformed into a structured format with 122 features. These include 38 continuous features and 84 binary features resulting from one-hot encoding of categorical attributes such as *protocol_type*, *flag*, and *service*.

The complete list of the 84 binary features derived from one-hot encoding is provided in Appendix A.3. In Table 4.15, we summarize the final set of features used for training and evaluation.

4.6 The CIC-IDS-2017 and CSE-CIC-IDS-2018 Datasets

To complement the NSL-KDD dataset and conduct realistic evaluations, we additionally use two modern datasets from the Canadian Institute for Cybersecurity (CIC): **CIC-IDS-2017** and **CSE-CIC-IDS-2018** [15]. Both datasets aim to reflect real-world network traffic and were created in controlled lab environments using contemporary operating systems, attack scripts, and traffic profiles.

CIC-IDS-2017 was collected over ten days and includes both benign and malicious traffic. It covers a wide spectrum of attacks such as DoS, DDoS, Brute Force, Heartbleed, Web attacks, Botnets, Infiltration, and Port Scanning. Traffic was captured using the CICFlowMeter tool, which extracted more than 80 features per flow. Ground truth labels were manually assigned, and timestamps enable session reconstruction. This dataset provides a high-fidelity simulation of enterprise network behavior and is widely used as a benchmark for modern intrusion detection.

CSE-CIC-IDS-2018 builds upon CIC-IDS-2017 by offering a broader and more refined coverage of attack vectors. It includes 14 attack types organized into 7 categories, such as Brute Force, Botnets, Web Attacks, and Infiltration. The traffic was captured using realistic scenarios executed over a period of ten days across multiple machines and users. As with its predecessor, CICFlowMeter was used to generate flow-based features. CSE-CIC-IDS-2018 addresses some of the limitations of the 2017 version by improving data labeling, balancing, and coverage of recent attack techniques.

Both datasets support robust benchmarking and generalization testing. In this thesis, we conduct inter-dataset evaluations: training on CIC-IDS-2017 and testing on CSE-CIC-IDS-2018, and vice versa. This setup allows us to analyze how well models trained on one dataset generalize to unseen traffic patterns and attack distributions — a key goal of this study, inspired by the recommendations in [21].

4.7 Preprocessing of the CIC-IDS-2017 and CSE-CIC-IDS-2018 Datasets

The CIC-IDS-2017 and CSE-CIC-IDS-2018 datasets were preprocessed to remove noise, and prepare the data for efficient training and evaluation. The following steps were applied consistently to both datasets and correspond to the preprocessing pipeline described

Table 4.15: Final Feature Set After Preprocessing of the NSL-KDD Dataset

Feature Name	Description
duration	Duration of the connection (in seconds)
src_bytes	Number of bytes sent from source to destination
dst_bytes	Number of bytes sent from destination to source
land_f	1 if the connection is from/to the same host/port; otherwise 0
wrong_fragment	Number of wrong fragments
urgent	Number of urgent packets
hot	Number of hot indicators
num_failed_logins	Number of failed login attempts
logged_in	1 if successfully logged in; otherwise 0
num_compromised	Number of compromised conditions
root_shell	1 if root shell is obtained; otherwise 0
su_attempted	1 if 'su root' command attempted; otherwise 0
num_root	Number of root accesses
num_file_creations	Number of file creation operations
num_shells	Number of shell prompts
num_access_files	Number of operations on access control files
num_outbound_cmds	Number of outbound commands in an FTP session
is_host_login	1 if the login belongs to the 'hot' list; otherwise 0
is_guest_login	1 if the login is a 'guest' login; otherwise 0
count	Number of connections to the same host in the past 2 seconds
srv_count	Number of connections to the same service in the past 2 seconds
serror_rate	% connections with SYN errors
srv_serror_rate	% connections to the same service with SYN errors
rerror_rate	% connections with REJ errors
srv_rerror_rate	% connections to the same service with REJ errors
same_srv_rate	% connections to the same service
diff_srv_rate	% connections to different services
srv_diff_host_rate	% connections to different hosts
dst_host_count	Number of connections to the same destination host
dst_host_srv_count	Number of connections to the same service on the destination host
dst_host_same_srv_rate	% connections to the same service on the destination host
dst_host_diff_srv_rate	% connections to different services on the destination host
dst_host_same_src_port_rate	% connections to the same source port on the destination host
dst_host_srv_diff_host_rate	% connections to different hosts on the same service
dst_host_serror_rate	% connections with SYN errors on the destination host
dst_host_srv_serror_rate	% connections to the same service with SYN errors on the destination host
dst_host_rerror_rate	% connections with REJ errors on the destination host
dst_host_srv_rerror_rate	% connections to the same service with REJ errors on the destination host

Note: The dataset contains a total of 122 features, including 38 continuous features listed above and 84 binary features derived from one-hot encoding. For a complete list of the 84 binary features, refer to Appendix A.3.

in the Towards Model Generalization for Intrusion Detection paper [21] and the GitHub repository of the author [20]:

1. **Column Harmonization:** Feature names across years were unified using a manually curated mapping. This step ensured consistency in feature naming conventions (e.g., renaming `TotLen Fwd Pkts` and `Total Length of Fwd Packets` to `Fwd Packets Length Total`).
2. **Removal of Non-Informative Features:** Several columns were dropped, including:
   - IP addresses and port numbers (`Source IP`, `Destination Port`, etc.)
   - Flags and metrics with no observed variance across samples (e.g., `Bwd PSH Flags`, `Fwd URG Flags`)
   - Duplicated features (e.g., `Fwd Header Length.1`)
3. **Timestamp Parsing and Sorting:** The `Timestamp` field was parsed and aligned to correct for inconsistent time zone shifts. All rows were sorted chronologically.
4. **Label Normalization:** The label `BENIGN` was mapped to `Benign` to match the expected case. Labels were then converted to categorical format.
5. **Data Type Corrections:** Integer and float fields were explicitly converted to reduce memory usage. Non-numeric values were coerced into NaNs and later removed.
6. **NaN and Infinity Handling:** Any rows containing NaN or infinite values were dropped to prevent downstream errors during training.
7. **Duplicate Removal:** Duplicate rows (excluding label and timestamp) were removed to ensure a cleaner dataset.
8. **Export to Multiple Formats:** The cleaned datasets were saved in `feather`, `csv`, and `parquet` formats to support efficient loading and compatibility with different frameworks.

This preprocessing pipeline ensured that the resulting data was clean, standardized across years, and free from spurious features. It formed the basis for further training and evaluation, described in Chapter 5. The resulting datasets have 67 features. The complete list of features used for training is provided in Appendix D.1.

5

Methods and Training Setup

In this chapter, we present the methods and training setup used in our experiments. We begin with a brief overview of the training setup which is originally based on the work of Caminero et. al. [4]. A detailed summary of the original AE-RL architecture is provided in Chapter 3. The setup is then extended to include new methodological contributions such as configurable attack types, architectural variations, multi-agent extension, and inter-dataset evaluation. The datasets and preprocessing steps applied to ensure data quality have been thoroughly described in Chapter 4.

5.1 Configurable Attack Type Selection

To enable systematic experimentation, the AE-RL framework was extended to support configurable attack type selection. This enhancement allows the training process to be restricted to specific categories of attacks—such as DoS, Probe, R2L, or U2R—based on the application scenario and the used dataset.

This functionality supports two training paradigms:

- **Binary classification:** Training a model to distinguish between benign traffic and a single attack type (e.g., DoS vs. Benign).
- **Multi-class classification:** Including multiple attack categories and benign traffic in the same training run (e.g., Normal, DoS, R2L, U2R).

This modular setup was crucial for conducting a wide range of experiments and evaluating the generalization capability of the defender agent across different types of attack patterns. It also forms the basis for further extensions such as inter-dataset evaluation and multi-agent attacker configurations.

5.2 Deep-Q-Learner Architecture

The Deep-Q-Learner architecture is based on the work of Caminero et al. [5] and the corresponding GitHub repository [4]. The deep neural network that approximates the Q-value function, is used to estimate the expected future rewards for each action taken in

a given state. The architecture is designed to be flexible and can be adapted to different datasets and attack types. Like in every Neural Network, the architecture consists of three main components: the input layer, hidden layers, and output layer.

1. The **input layer** accepts observations from the environment, with a dimensionality corresponding to the number of features in the dataset. These states represent the current network traffic data, including both normal and attack samples.

2. The **hidden layers** are fully connected and the amount can be configured via the `hidden_layers` parameter. Each hidden layer has `hidden_size` neurons, and utilizes the ReLU activation function to introduce non-linearity into the model.

3. The **output layer** (Dense layer) has `num_actions` neurons, corresponding to the number of possible actions of an agent e.g., selecting an attack in case of the attacker, or classifying a sample in case of the defender. The output layer uses a linear activation function to produce Q-values for each action. The Q-values represent the expected future rewards for each action given the current state.

4. The model utilizes the **Adam optimizer** with a learning rate of `learning_rate` to update the weights during training. The loss function is the **Huber loss**, which is less sensitive to outliers than the mean squared error loss.

Additional metrics such as Mean-Squared-Error (MSE), Mean-Absolute-Error (MAE) are monitored during training to evaluate the model's convergence and performance. Recall, Precision, Accuracy and AUC calculated by the in-built TensorFlow metrics, can not be interpreted correctly due to the Q-learning environment. The Q-Values are not directly comparable to the labels. Therefore, these metrics are calculated separately after the Learning phase during the consecutive test phase.

Both, the Attacker and Defender agents are implemented using the same *input architecture*, which in case of the NSL-KDD dataset consists of 122 features in case of CIC-IDS-2017 and CSE-CIC-IDS-2018 67 features.

For the Attacker, the authors of [5] used one hidden layer with 100 neurons. The output layer has in case of NSL-KDD 23 neurons, which correspond to each possible attack of all attack types ('Normal', 'DoS', 'Probe', 'R2L' and 'U2R' details in A.3).

On the other hand the Defender agent uses 3 hidden layers with 100 neurons each. The output layer has 5 neurons, which correspond to the five attack classes of the NSL-KDD dataset.

Caminero et. al. [5] reasoned, that the '… simplicity of this architecture provides a competing response time while the reinforcement learning training optimally adjusts weights and biases'. Beside the fast response time, no further reasons were given for the choice of the architecture. Therefore, we decided to make an empirical evaluation of the architecture. The Deep-Q-Learner are implemented using the Keras library with TensorFlow as the backend.

5.3 Training Setup

Learning Rate Selection

The authors of [5] did specify using a learning rate of 0.001 during training, but did not provide any further information about it. To evaluate the impact of different learning rates on model performance, several values were tested.

Experiments were conducted with two architectural configurations: (1) an Attacker with 5 hidden layers and a Defender with 3 hidden layers, each comprising 100 neurons; and (2) an Attacker with 1 hidden layer and a Defender with 3 hidden layers (also with 100 neurons per layer). Each configuration was trained using learning rates in the range $[0.00025, 0.0025]$, and each experiment was repeated three times to calculate average performance scores.

Training followed the original setup by Caminero et al. [5], using 100 episodes with 100 iterations each.

Across both configurations, a learning rate of 0.001 consistently yielded the best results in terms of average accuracy and F1 score. This value also coincides with the default learning rate of the Adam optimizer in Keras. A complete overview of all results is available in a publicly accessible spreadsheet[1].

Using a learning rate of **0.001** resulted in the best performances across many configuration, however the performance is not significantly different and varies between 1-3% in terms of accuracy and F1 score.

5.4 Feature Selection via Boruta

Motivation and Introduction

As the authors of the original work [5] did not provide details on their feature selection strategy for the NSL-KDD and AWID datasets and instead trained on all available features, we employed the Boruta algorithm to identify the most relevant features for the NSL-KDD dataset and assess model performance using only these.

The Boruta algorithm is a wrapper method for feature selection that uses a random forest classifier to determine the importance of each feature. It compares the importance of each feature to that of shadow features, which are copies of the original features that are randomly shuffled to achieve noise. If a feature is more important than the maximal important shadow features, it is considered relevant and kept for the final feature set. The Boruta algorithm is designed to handle high-dimensional data sets with many features and is particularly useful for identifying the most important features in complex data sets. More specific Boruta is solving the *all-relevant* feature selection problem, which means it tries to find all features carrying information usable for prediction, rather than finding a minimal subset of features on which some classifier has good performance.

After applying the Boruta algorithm on a dataset, each of the present features get one of the following status: 'confirmed', 'rejected' or 'tentative'. In case tentative features are

[1] https://docs.google.com/spreadsheets/d/1Uak1kR54UffO9Q2jjN1uvLxOFaKkG0_76U_Fm_ayMg8/edit?usp=sharing

present, the boruta package provides the *TentativeRoughFix* method which decides automatically a status of 'confirmed' or 'rejected' for the given feature. Detailed information about the Boruta algorithm can be found in the original paper [10].

Robustness of Feature Selection with Boruta

Due to the stochastic nature of the Boruta algorithm (which relies on Random Forests), it is important to evaluate the stability of the selected features across multiple runs. In our experiments, we executed the Boruta selection procedure three times on the same dataset, each time with a different random seed.

While the resulting importance scores showed minor variations, the set of confirmed important features remained largely consistent across all runs. This indicates that the feature selection process is robust and not overly sensitive to initialization or minor sampling differences. Consequently, we can rely on the selected features for downstream model training and evaluation with a high degree of confidence.

Subset Definitions: Random, Stratified, Balanced, Manually Filtered

Since using the whole NSL-KDD training set was impracticable with my personal hardware, four subsets of data were used instead to run the Boruta algorithm on these subsets.

First of all a subset of *randomly* 10% of samples were chosen. In addition a *stratified* subset of 10% of data was chosen by using the 'createDataPartition' method of the 'caret' package in R. Both, the randomly chosen data and the stratified data received by 'caret' have almost the same distribution (see 5.1) for the chosen attacks of each attack type 'Normal', 'DoS', 'Probe', 'R2L' and 'U2R'.

I call the third subset the *balanced* one. It is composed of using all 995 'R2L' and 52 'U2R' samples. The remaining three attack types 'Normal', 'DoS' and 'Probe' each contain 3885 samples. In total the *balanced* set contains 12702 samples which is about 10% of the whole NSL-KDD train set.

Table 5.1: Comparison of dataset subsets used for Boruta analysis (sample sizes per attack type).

Subset	**Normal**	**DoS**	**Probe**	**R2L**	**U2R**
Random (≈ 10%)	6794	4499	1199	102	4
Stratified (≈ 10%)	6703	4624	1167	100	4
Balanced (≈ 10%)	3885	3885	3885	995	52
Manually Filtered (≈ 10%)	6687	4640	1163	103	5

The rationale behind including all samples of R2L and U2R is based on the assumption that certain discriminative patterns may only emerge when a sufficient number of examples from these under-represented classes are available. Without enough samples, important features might be overlooked by the underlying random forest algorithm due to insufficient variance and representation.

In addition to the three described subsets, another dataset was created based on a manual filtering of highly correlated features. This dataset will hereafter be referred to as the *Manually Filtered Dataset*.

The creation of this dataset involved analysing the correlations between features in the training dataset. Features with a high correlation (threshold: > 0.9) were removed, as they probably contain redundant information and could negatively impact model performance. The filtering process was conducted using the *findCorrelation* function from the *caret* package in R.

Additionally, the *num_outbound_cmds* feature was removed because it contains only zero values and does not contribute any predictive information. The filtering process was performed iteratively across three passes which is described in subsection Correlation Matrix Analysis. After three runs of filtering, 113 features remained in the dataset, which served as the basis for the Boruta analysis. The goal of this manual filtering was to reduce the dimensionality of the dataset and improve the interpretability of the results without losing relevant information.

Correlation Matrix Analysis

Summary of Correlation Filtering: A correlation-based filtering removed 8 features over three iterations using a Pearson threshold of > 0.9. Table 5.2 lists the affected features and their strongest correlation partner. This step helped reduce multicollinearity and redundancy in the dataset. In addition, `num_outbound_cmds` was excluded due to a constant value across all records.

As outlined in the summary above, the correlation filtering was performed iteratively based on a Pearson threshold of > 0.9. Table 5.2 provides a detailed breakdown of each step.

Table 5.2: Features removed during correlation filtering (threshold > 0.9).

Iteration	Removed Feature	Correlation Partner (r)
1	srv_serror_rate	serror_rate (0.9933)
	srv_rerror_rate	rerror_rate (0.9890)
	dst_host_srv_serror_rate	srv_serror_rate (0.9863)
	dst_host_srv_rerror_rate	srv_rerror_rate (0.9702)
2	dst_host_serror_rate	serror_rate (0.9794)
	dst_host_rerror_rate	rerror_rate (0.9267)
	S0	serror_rate (0.9792)
3	num_compromised	num_root (0.9988)

After three iterations, no features with a correlation coefficient greater than 0.9 remained in the dataset.

In total, the following 9 features were removed during the process:

- *srv_serror_rate*, *srv_rerror_rate*, *dst_host_srv_serror_rate*, *dst_host_srv_rerror_rate*
- *dst_host_serror_rate*, *dst_host_rerror_rate*, *S0*, *num_compromised*
- *num_outbound_cmds* (constant value of 0)

This iterative process reduced redundancy in the dataset and ensured that the remaining features were less likely to introduce noise or bias into the subsequent analyses.

Attack-Specific Feature Relevance Analysis

To identify features that are most relevant for distinguishing specific attack types, the Boruta algorithm was applied to separate binary classification tasks. Each task consisted of one attack category (*DoS*, *Probe*, *R2L*, or *U2R*) combined with normal traffic.

Motivation and Assumption: It is assumed that different types of attacks exhibit distinct behavioural patterns and network characteristics. These behavioural differences should be reflected in distinct sets of relevant features, e.g., login attempts for *R2L*, or privilege escalation indicators for *U2R*. This assumption is grounded in the nature of the attack objectives and execution paths. For instance, a Denial of Service (*DoS*) attack typically involves overwhelming a target host with a high number of requests, leading to distinctive patterns in connection count and packet rates. In contrast, a *Probe* attack aims to gather information about open ports and services on a target host, which closely resembles normal behaviour in some scenarios (e.g., during software development or routine maintenance), making the discrimination more subtle.

Remote to Local (*R2L*) attacks require a more complex sequence of actions: they begin by exploiting a remote service, then attempt to gain unauthorized local access by exploiting login mechanisms or known vulnerabilities. These attacks are likely to be reflected in features related to failed login attempts, shell invocations, or suspicious command executions.

User to Root (*U2R*) attacks go even further by exploiting privilege escalation vulnerabilities after a user has already gained local access. This may involve advanced system interactions, such as the use of 'setuid' binaries, kernel exploits, or other techniques aimed at acquiring root access. As a result, they activate a distinct set of low-frequency system features.

Methodology:

1. For each attack category, up to 2000 samples were selected and combined with 2000 samples of normal traffic.
2. The Boruta algorithm was executed on this dataset, using the attack category as the target label.
3. Tentative features were resolved using the `TentativeRoughFix` method.
4. The resulting feature importances were stored and compared across categories.

Results and Observations: The Boruta analysis revealed partially disjoint sets of relevant features for each attack category:

- *R2L* and *U2R* attacks were primarily associated with features related to user authentication and privilege escalation (e.g., `root_shell`, `login`, `num_file_creations`).
- *DoS* and *Probe* attacks were associated with features characterizing traffic behavior (e.g., `urp_i`, `RSTOS0`, `RSTR`, `S0`, `S1`).

This experiment confirms that attack-specific feature selection is a valuable strategy for designing interpretable and specialized detection models. By isolating the most relevant features for each category, this approach enhances the interpretability of intrusion

detection models and provides valuable insights into the underlying characteristics of different attack behaviors. A detailed overview of the features determined as confirmed by the Boruta algorithm is provided in subsection Comparative Feature Overlap Analysis.

Comparative Feature Overlap Analysis

To evaluate the consistency and generalizability of the selected features, we performed a comparative overlap analysis of Boruta confirmed features across multiple dataset variants. These include all previously described sampling strategies (random, stratified, balanced), correlation-based filtering, and attack-specific subsets (Normal vs. DoS, Probe, R2L, U2R).

Core Feature Stability: The intersection of confirmed features across all subsets (including attack-specific ones) revealed a stable core of **24 features**, such as *src_bytes*, *dst_bytes*, *count*, *srv_count*, *same_srv_rate*, and *SF*. These attributes were consistently marked as relevant regardless of sampling method or attack type, underscoring their fundamental role in differentiating normal from malicious behavior.

When limiting the comparison to the four general subsets (random, stratified, balanced, manually filtered highly correlated features) the overlap increases to **42 confirmed features**. This suggests that attack-specific subsets reveal more specialized but less generalizable features, complementing those found through generic sampling strategies.

Sampling Strategy Effects: A detailed comparison between stratified and balanced subsets shows that **11 features** differ in relevance. Notably:

- *IRC* and *S1* were confirmed exclusively in the *stratified* subset as relevant.
- *imap4*, *num_file_creations*, *num_shells*, *num_access_files*, *auth*, *discard*, *login*, *time* and *uucp* are only confirmed in the *balanced* variant.

This indicates that balancing the dataset enhances the detectability of low-frequency class-specific features, especially for under-represented attack types such as R2L and U2R. Conversely, stratified or unbalanced sampling favours features dominant in majority classes like DoS.

Single-Attack-Type Specificity: Additionally, our analysis shows that the feature *urgent* was marked as relevant exclusively in the Normal vs. R2L subset. This highlights that attack-specific evaluations can expose specialized signals that are otherwise overshadowed in multi-class or imbalanced settings.

Summary: The overlap analysis emphasizes the importance of combining general sampling strategies with targeted attack-specific feature selection. It ensures both the identification of robust, general-purpose features and the detection of specialized indicators critical for less frequent intrusion types.

To gain more insights into the results we visualized the results of confirmed, rejected and tentative features in distinct figures. The following figure 5.3 contains a representation of all confirmed features. The features that were determined as 'rejected' (E.2) and 'tentative' (E.3) can be found in the appendix E.

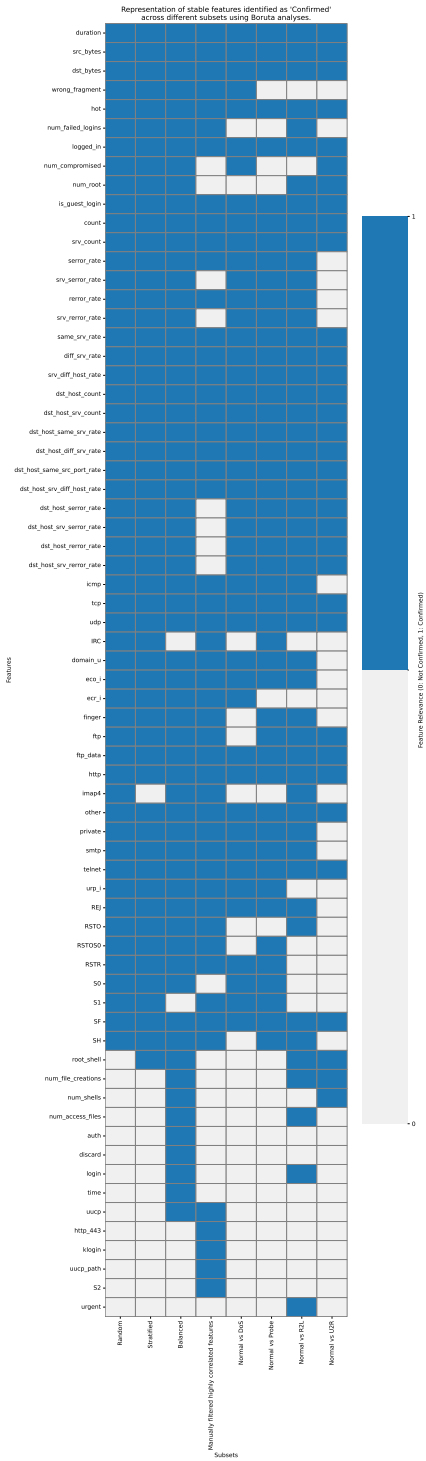

Figure 5.3: Confirmed relevant features determined by the Boruta algorithm across different datasets. The X-axis represents the datasets on which the Boruta algorithm was executed, while the Y-axis lists the feature names. Blue-filled cells indicate relevant 'Confirmed' features, and gray-filled cells represent 'Not Confirmed' features.

5.5 Analysis of the Computation Power of the Q-Networks

We have used different configuration for the Q-Networks of the Attacker and Defender agents. These configuration can be divided into three categories:

1. The Attacker agent has more computation power in terms of the number of hidden layers and neurons compared to the Defender agent.
2. The Defender agent has more computation power in terms of the number of hidden layers and neurons compared to the Attacker agent.
3. The Defender agent has the same computation power in terms of the number of hidden layers and neurons compared to the Attacker agent.

The same learning rate of 0.001 was used for all consecutive experiments because it was the best performing learning rate measured. More details of the performance of the different configurations can be found within Experiment 2: Comparison of Computational Power.

5.6 Extending the Framework to Support Multiple Attackers

This work extends the original AE-RL[5] framework to support multiple agents, specifically multiple attacker agents and a single defender agent. However, the extension of the framework to support multiple defender agents has been considered but not fully implemented nor evaluated in this thesis and is an interesting subject for future work.

The original AE-RL framework was designed for a single attacker agent and a single defender agent. In this setup, the attacker agent selects one sample (either normal or attack) from the dataset, which is then presented to the defender agent. The defender agent processes this sample. The environment provides feedback to both agents, which in turn updates their policies based on the received reward. In case the the defender agent correctly classifies the sample chosen by the attacker agent, the defender gets rewarded with +1. The attacker gets no reward in this case. In case the defender agent misclassifies the sample chosen by the attacker agent, the attacker gets rewarded with +1 and the defender gets no reward instead.

While extending the framework to support multiple attacker agents, we decided to keep the same reward structure but to confront the defender agent with multiple samples per iteration per episode. Each of the attacker agents selects one sample from the dataset, which is then presented to the defender agent. That means that the defender takes actions based on the 4 samples chosen by the 4 attacker agents. The defender can receive a reward of up to +4 in case all 4 samples are classified correctly. On the other hand, each attacker agent can receive a reward of +1 in case the defender misclassifies the sample chosen by the corresponding attacker agent.

5.7 Asymmetric Experience Replay between Multiple Attackers and a Single Defender

In the previous framework, experiences were collected symmetrically: one attacker agent selected one sample (either normal or attack), which was subsequently presented to the defender agent. Consequently, both the attacker and the defender accumulated experiences at an identical rate, resulting in equally sized Replay Memories. Each Replay Memory, with a capacity of 1000 samples, enabled consistent training conditions for both agents.

In the newly proposed extension, however, we introduce four attacker agents, each independently selecting one sample per iteration. This modification leads to the defender being confronted with four samples per iteration, in contrast to a single sample per attacker. Hence, the defender accumulates experiences four times faster than any individual attacker, creating an asymmetry in memory requirements and learning dynamics.

Problem Description

Given the unchanged Replay Memory capacity of 1000 samples, the defender's Replay Memory becomes saturated after only approximately 2.5 episodes (each episode consisting of 100 iterations, thus yielding 400 samples per episode for the defender). This rapid turnover significantly reduces the retention of historical data, possibly leading to a loss of valuable older experiences that might be crucial, especially in highly imbalanced anomaly detection scenarios.

Potential Solutions

To address this imbalance, we considered three possible approaches:

1. **Increase Replay Memory capacity for the defender by a factor of four (to 4000 samples)**: This ensures that both attackers and defender store experiences over a similar duration (approximately 10 episodes).
2. **Maintain the Replay Memory size but increase the defender's minibatch size during training (e.g., from 100 to 400 samples)**: This enables faster learning from the available experiences but causes rapid memory turnover, potentially losing important historical data.
3. **Combine both methods—enlarge the Replay Memory and moderately increase the minibatch size for the defender (recommended)**: This option provides the advantages of both approaches: improved historical experience retention due to increased memory size, and enhanced learning effectiveness via a larger minibatch size.

Recommended Approach and Justification

We identified the third option as the most suitable solution and the extra resources required for the defender's Replay Memory are justified by the increased computational power of modern systems. This approach effectively balances the need for historical experience retention with the requirement for efficient learning. Specifically, increasing

the defender's Replay Memory capacity to 4000 samples and moderately adjusting its minibatch size to 200 ensures the following benefits:

- **Improved memory retention**: By quadrupling the Replay Memory size, the defender retains critical historical experiences, which are essential for stable and robust learning, especially in imbalanced scenarios.
- **Enhanced training efficiency**: Increasing the minibatch size facilitates more effective updates of the defender's model parameters, as each training iteration encompasses a broader and more diverse set of experiences.
- **Balanced resource usage**: This combined approach maintains computational efficiency and manageable memory requirements while significantly improving the defender's ability to learn from its experiences.

Consequently, the third option provides a balanced, robust, and efficient strategy to effectively handle the inherent asymmetry in experience accumulation within the extended multi-attacker-single-defender scenario.

5.8 Training Setup for CIC-IDS Datasets and Methodological Adaptations

After preprocessing, the CIC-IDS-2017 and CSE-CIC-IDS-2018 datasets are loaded via a dedicated data management module. This module supports configurable training setups, including binary and multi-class classification as well as inter- and intra-dataset evaluations.

Motivation and Alignment with Related Work

To allow for meaningful comparison, this work follows the methodology of Verkerken et al. [21], who investigated anomaly detection in the CIC-IDS datasets using unsupervised, binary classifiers. While their models were trained only on benign samples, our reinforcement learning (RL) framework requires both benign and malicious examples to train the defender agent to differentiate between normal and attack traffic.

Verkerken et al. explicitly called for future work adapting their evaluation setup to supervised learning. This thesis addresses that gap by translating their binary evaluation structure into a (semi-)supervised RL environment, while also supporting multi-class classification to reflect more complex detection scenarios.

Dataset Splitting and Sampling Strategy

To mirror the original conditions used by Verkerken et al., we adopt their data splitting strategy:

- **CIC-IDS-2017:** Benign samples are downsampled to 250,000. Of these, 50,000 are used for training. The remaining 200,000 benign samples and all attack samples are stratified into test and validation sets in a 70:30 ratio.

– **CSE-CIC-IDS-2018:** Benign samples are downsampled to 550,000. Again, 50,000 are used for training, with the remaining 500,000 and all attack samples split 85:15 into test and validation sets.

Class Ratio Preservation for Training

To preserve the natural imbalance of the original datasets, we calculate the number of attack samples to include in the training set according to the original class distribution ratios (Table 5.4). The formula used is:

$$attack\_train\_count = \left\lceil train\_benign\_size \cdot \frac{malicious\_ratio}{benign\_ratio} \right\rceil$$

For instance, when training on CIC-IDS-2017 with 50,000 benign samples and a DoS ratio of 13.43% versus 80.32% benign, we get:

$$attack\_train\_count = \left\lceil 50{,}000 \cdot \frac{0.1343}{0.8032} \right\rceil = 8{,}360$$

Thus, the training set contains 50,000 benign and 8,360 DoS samples in this binary scenario.

Support for Binary and Multi-Class Training

Our training pipeline supports:

– **Binary classification:** The training set consists of benign samples and samples from a single attack type (e.g., DoS).
– **Multi-class classification:** Multiple attack types are included according to their original ratios in the dataset.

This flexibility allows for controlled evaluation under both simplified and realistic threat scenarios.

Dataset Statistics

Implementation Notes

The `CICDataManager` module handles all relevant steps, including:

– Dataset loading and preprocessing,
– Downsampling of benign traffic,
– Attack sample selection based on the ratios in Table 5.4,
– Train/validation/test splitting, and
– Setup for intra- and inter-dataset evaluation.

The module exposes a simple interface to configure these options dynamically.

Table 5.4: Class distribution ratios of attack types in CIC-IDS datasets.

Attack Type	CIC-IDS-2017	CSE-CIC-IDS-2018
Benign	80.32%	84.59%
(D)DoS	13.43%	11.17%
Probe	5.62%	0.00%
Brute Force	0.49%	1.08%
Web Attack	0.08%	0.10%
Botnet	0.07%	1.66%
Infiltration	0.01%	1.49%
Heartbleed	0.01%	0.00%

5.9 Evaluation Strategy

To assess the generalization capabilities of the defender agent, both intra- and inter-dataset evaluations were conducted:

- **Intra-dataset evaluation:** Training and testing are performed on different splits of the same dataset.
- **Inter-dataset evaluation:** The model is trained on CIC-IDS-2017 and tested on CSE-CIC-IDS-2018, and vice versa.

This design follows the setup of Verkerken et al. [21], who used this approach to evaluate anomaly detectors generalizability across datasets with differing traffic distributions. In our work, it enables a fair comparison while adapting the evaluation to a supervised RL setting.

As described in Section 5.8, the `CICDataManager` supports both evaluation modes. Standard performance metrics such as accuracy, F1-score, precision, and recall are computed post-training to quantify the defender's ability to generalize across known and previously unseen attack types.

5.10 Summary: Framework Extensions and Architectural Modifications

As part of this thesis, we extended the AE-RL framework originally proposed by Caminero et al. [5] to support more flexible training scenarios, refined agent architectures, and realistic anomaly detection conditions. These modifications were necessary both to adapt the framework to modern datasets and to enable rigorous evaluation under varied and realistic conditions.

Configurable Attack Type Selection

The AE-RL codebase was modified to allow attacker agents to be limited to specific attack types. This enables precise control over training samples by filtering attacks according to high-level attack categories (e.g., DoS, Probe, R2L, U2R). This modification lays the

groundwork for both binary setups (e.g., benign vs. DoS) and more complex multi-class (easy extensible to one-vs-all) scenarios, and ensures that the model learns under interpretable and reproducible conditions.

Multi-Agent Architecture and Asymmetric Experience Accumulation

To better simulate realistic threat scenarios, the AE-RL framework was extended to support multiple attacker agents operating in parallel. This multi-agent setup introduces an asymmetry in experience accumulation, as discussed in detail in Section 5.6 and Section 5.7. To address this, the defender uses a larger replay memory and an adjusted mini-batch size.

Support for Inter-Dataset Evaluation

Inspired by Verkerken et al. [21], we extended the AE-RL framework to support inter-dataset evaluation. In this setup, agents are trained on one dataset (e.g., CIC-IDS-2017) and tested on another (e.g., CSE-CIC-IDS-2018), and vice versa.

This functionality enables rigorous testing of a model's ability to generalize to unseen traffic distributions and attack types—a key aspect of real-world intrusion detection. To align with Verkerken et al.'s binary anomaly detection design and to remain as close as possible to their experimental setup, we adapted the training and evaluation processes to reflect a binary classification paradigm. Additionally, we ensure that the attack distributions in both datasets are preserved, allowing for a fair comparison of model performance.

In the following chapter, we empirically evaluate the contributions discussed above.

6

Experiments and Results

To gain deeper insights into the behavior of the original model proposed by Caminero et al. [5], we decided to divide the training data into multiple subsets and train the model separately on each of them. This allows to systematically analyze how different training data compositions affect the model's performance. The results of these experiments are presented in Section 6.2.

Since the original work [5] provides no detailed explanation for the architectural design choices—apart from stating that the defender agent was equipped with three hidden layers to ensure fast response times for the intrusion detection system—we conducted an empirical investigation of the model using various configurations. Specifically, we varied the number of hidden layers for both the attacker and defender agents and compared the resulting performance across these configurations. More details on the results of these experiments can be found in Section 6.3.

We also extended the original framework to support multiple attacker agents and asymmetric experience replay. This extension allows the model to simulate more realistic scenarios where multiple attackers operate simultaneously, each selecting samples independently. The results of these experiments are presented in Section 6.5.

Finally, we conducted a feature selection analysis using the Boruta algorithm to identify the most relevant features for the model. This analysis was performed on the NSL-KDD dataset, and the results are presented in Section 6.6.

In addition to the experiments on the NSL-KDD dataset, we also evaluated the model on two more recent datasets: CIC-IDS-2017 and CSE-CIC-IDS-2018. These datasets were preprocessed and prepared for training, as described in Chapter 4. The results of these experiments are presented in Section 6.7.

All performance metrics reported in this chapter—such as accuracy, precision, recall, and F1-score—were calculated using the `scikit-learn` library [12]. Unless otherwise noted, we report weighted averages for these metrics to account for class imbalance in the test sets. This means that each class contributes to the overall score in proportion to its support (i.e., the number of true instances for that class), providing a more realistic evaluation in scenarios with heavily skewed class distributions, such as intrusion detection. In addition all models were trained with a *learning rate of 0.001* unless otherwise specified.

6.1 Impact of Class Imbalance on Precision and Recall

One of the key challenges in intrusion detection is the highly imbalanced nature of network traffic datasets, where benign samples vastly outnumber malicious ones. This imbalance affects the learning dynamics of the AE-RL framework, particularly in how the defender agent optimizes its decision boundaries.

In such settings, models tend to achieve high overall accuracy by favouring the majority class (typically 'Normal'), but this often leads to low recall for rare attack types such as R2L or U2R. Precision, on the other hand, may appear high due to a low number of false positives—but this is misleading if true positive detection is poor.

To evaluate this effect, we conducted experiments with different data splits: from training on all available classes (highly imbalanced), to training on a balanced subset of normal and anomalous samples, as well as training exclusively on selected attack types. We also introduced a synthetically balanced dataset where under-represented attack classes were upsampled. The results show that reducing class imbalance improves recall for rare attacks, but may slightly lower precision due to an increase in false positives.

These trade-offs are analysed in detail in the following experiments.

6.2 Experiment 1: NSL-KDD - Impact of training on different Data Subsets

During training, the attacker agent was restricted to select only one or several specific classes. This constraint did not apply to the defender agent, which retained the ability to classify each presented sample into one of five classes: `Normal`, `DoS`, `Probe`, `R2L`, and `U2R`.

The model was trained on different datasets as well as on subsets of specific attack types within a dataset. This approach allowed for a detailed evaluation of the model's performance under varying conditions, including scenarios with limited attack diversity and imbalanced class distributions. The following training setups were considered:

1. Training on a single class.
2. Training exclusively on the attack classes `DoS`, `Probe`, `R2L`, and `U2R`.
3. Training on two classes Normal & <AttackType> (e.g., `Normal` and `DoS`).
4. Training on a set with the same amount of normal and anomalous data. For this setup, the number of normal samples was reduced from 67,343 to match the number of anomalous samples (58,630).
5. Training on the balanced dataset described in Section 4.3.

1. Training on a Single Class

When training on a single class, the model fails to generalize because it is only exposed to one specific class during training. As a result, the model achieves 100% true positives for the trained class, but this is solely because it always predicts that one class. Consequently, the corresponding column in the confusion matrix is entirely filled with ones.

Since the results for the 'Normal' class were identical across repeated runs, training on the classes DoS, Probe, R2L, and U2R were conducted once without averaging over multiple runs.

The confusion matrix for the U2R class shown in Figure6.1 represents the actual class occurrences on the Y-axis and the predicted classes on the X-axis. All values are normalized between 0 and 1 for all confusion matrices depicted in the experiments. As expected, the entire column for the U2R class is filled with 1, as it is the only class predicted by the model. The remaining confusion matrices for other classes are included in the AppendixC for the sake of completeness.

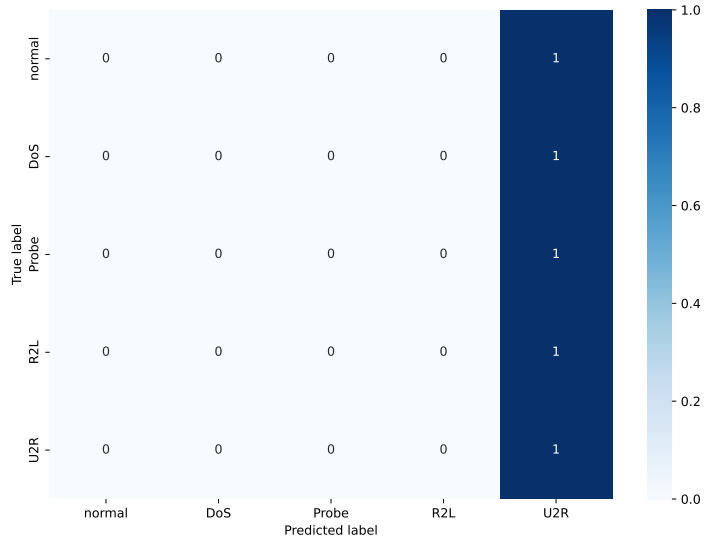

Figure 6.1: Confusion matrix for training on a single class "U2R". Since it is the only class present during training, the model does not learn to generalize, resulting in the entire column for the class "U2R" being filled with 1.

2. Training exclusively on attack classes:
When training exclusively on the attack classes DoS, Probe, R2L, and U2R, the resulting model achieves good True Positive (TP) rates for the individual attacks. The achieved TP results are comparable to a model trained on all classes, with the R2L class being detected better than in the model trained on all classes.

For comparison, we examine the confusion matrix of the *All* model, which achieved the best results so far, shown in Figure 6.6. In contrast, the *Attacks* model, which was trained only on the attack classes DoS, Probe, R2L, and U2R, is shown in Figure 6.7.

It can be observed that the *Attacks* model detects the attack class R2L significantly better, achieving a TP rate of 0.745, compared to the *All* model, which achieves only 0.286. For the remaining attack classes, the *Attacks* model performs similarly well to the *All* model some classes are better detected, while others are worse. All confusion matrices of the trained *Attacks* models are included in the Appendix C for completeness.

3. Training on Two Classes, e.g., 'Normal' and 'DoS'

When training on two classes, such as 'Normal' and 'DoS,' the model learns exclusively to distinguish between these two classes. For example, such a model achieves a True Positive (TP) rate of 0.980 for the 'Normal' class and 0.842 for the 'DoS' class, as shown in Figure 6.8.

Comparing these TP rates with the *All* and *Attacks* models reveals that the detection performance for the 'Normal' class is better than that of the *All* model, while the detection rate for the 'DoS' class is worse than that of the *Attacks* model.

Nevertheless, two-class models perform worse overall compared to models trained on all classes or only on the attack classes 'DoS,' 'Probe,' 'R2L,' and 'U2R.'
To systematically analyze the differences between these models, various metrics were considered, including Accuracy, Precision, Recall, and F1-Score in a One-vs-Rest comparison. This means that each metric was calculated for a single class, while all other classes were treated as the opposing category. This allows for a detailed evaluation of the model's performance for each attack class in isolation and follows the approach of Caminero et al. [5].

We show in Figure 6.2, three model variants:

1. The *All* model, trained on all classes.
2. The *Attacks* model, trained only on the attack classes 'DoS', 'Probe', 'R2L', and 'U2R'.
3. The *Normal & Class-X* models, where a separate model was trained for each attack class. The data used included the 'Normal' class and one specific attack class (e.g. 'Normal' & 'DoS', 'Normal' & 'Probe', etc.).

It is important to note that regarding the *Normal & Class-X* models, the X-axis contains multiple models (one model) for each attack category. The Y-axis shows the corresponding One-vs-All metric values, enabling direct comparisons between the different training approaches. Additionally, it should be noted that the metrics for the 'Normal' class in the *Attacks* and *Normal & Class-X* models are not directly comparable or meaningful.

In the case of the 'Normal & Class-X' models, the performance for the 'Normal' class strongly depends on the frequency of the respective attack class. This leads to significant variations in metrics such as the F1-score depending on the attack class. For example, the model for 'Normal & DoS' achieves an F1-score of 0.7272, while the model for 'Normal & U2R' achieves only 0.6137. These differences result from the varying distribution of classes in the training dataset and thus influence the model evaluation. In contrast, the 'Attacks' model does not include the 'Normal' class in the training. Therefore, no meaningful model behavior exists for this class, making a comparison of metrics such as Accuracy, F1-score, Recall, and Precision in this case not meaningful.

The results show that the 'All' model, trained on all classes, achieves the best overall results. However, an exception is the attack class 'R2L', which is particularly well detected by the 'Attacks' model. This suggests that it may be advantageous for the 'R2L' class to train the model exclusively on the attack classes 'DoS', 'Probe', 'R2L', and 'U2R'.

One possible reason for the differing classification performances could be the number of available training instances. In the 'All' model, the 'R2L' class is relatively rare (see Figure 6.9), while it is overrepresented in the 'Normal & R2L' model (see Figure 6.11). The

'Attacks' model, on the other hand, appears to have a more balanced distribution (see Figure 6.10), which may lead to better classification performance for 'R2L'.

To better illustrate the development of class distribution during training, nine snapshots were created for different epochs just like in the original work [5]. These can be viewed as a kind of time-lapse to visualize the changes in distribution over the training period.

Examining the distribution of training instances in the 'Normal & R2L' model reveals that the 'R2L' class is significantly overrepresented compared to the other attack classes. During training, around 700 instances of this class were processed. In comparison, the 'Attacks' model processed only about 300 instances of the 'R2L' class, while the 'All' model processed only around 260 instances.

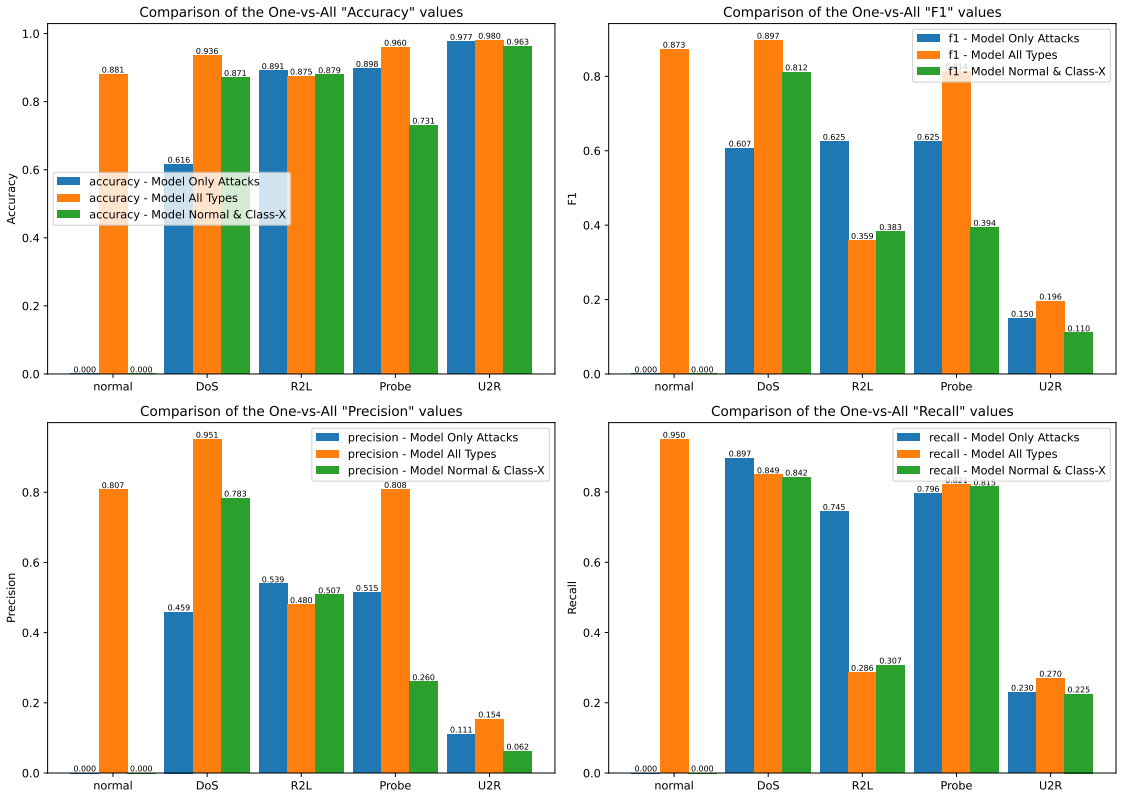

Figure 6.2: Comparison of various One-vs-All metrics for the models *'All'*, *'Attacks'*, *and 'Normal & Class-X'*.
The *All* model was trained on all sample classes.
The *Attacks* model was trained only on the attack classes 'DoS', 'Probe', 'R2L', and 'U2R'.
The *Normal & Class-X* model refers to a training strategy where a separate model was trained for each attack class, including only 'Normal' and a specific attack class (e.g., one model for 'Normal' & 'DoS', another for 'Normal' & 'Probe', etc.). Therefore, the X-axis segments for 'Normal & Class-X' consist of multiple individual models. The Y-axis shows the values for Accuracy, Precision, Recall, and F1-Score.

4. Training on an equally distributed set of normal and anomalous data.

Another approach to analyze the model's performance involved adjusting the training to an equally distributed set of normal and anomalous data. For this purpose, the number of normal instances was reduced from originally 67,343 to 58,630, matching the total number of anomalous instances.

However, it became evident that this reduction did not lead to any performance improvement. On the contrary, the test results suggest that reducing the number of normal instances during training can even be detrimental. To verify this, three model training sessions were conducted with different datasets. The results consistently show that artificially balancing the data by removing normal instances is not beneficial. Instead, a larger number of normal training instances appears to be important for the model's performance.

Table 6.3 lists the results for trained models without downsampling the normal instances, while Table 6.4 shows results for models trained on an equally distributed set of normal and anomalous data. On average, models without artificial reduction of normal instances achieve better results in metrics such as Accuracy, F1, Precision, and Recall compared to models trained on an equally distributed set of normal and anomalous data.

Table 6.3: Results for trained models without downsampling normal instances.

	First Run	Second Run	Third Run	Average
Accuracy	0,8376	0,7896	0,7992	0,8088
F1	0,8357	0,7787	0,7953	0,8032
Precision_score	0,8400	0,7808	0,7932	0,8047
Recall_score	0,8376	0,7896	0,7992	0,8088

Table 6.4: Results for models trained on *equally balanced data* (Downsampled 'Normal' to 58,630 instances). Hint: This dataset differs from the *balanced dataset* described in Section 4.3, as it does not include adjustments to the R2L and U2R attack distributions.

	First Run	Second Run	Third Run	Average
Accuracy	0,7943	0,7886	0,7863	0,7897
F1	0,7848	0,7738	0,7797	0,7794
Precision_score	0,7883	0,7774	0,7797	0,7818
Recall_score	0,7943	0,7886	0,7863	0,7897

5. Training on the balanced dataset

The last training setup involved training on the *balanced dataset* described in Section 4.3. Results for this training setup are aggregated in Table 6.5. The rebalanced dataset mitigates the effects of class imbalance, allowing the model to learn more representative decision boundaries, particularly for under-represented classes.

On average, models achieve best performance metrics for the NSL-KDD dataset, with weighted average metrics on accuracy of 0.8479, an F1-score of 0.8561, a precision score of 0.8703, and a recall score of 0.8479. These results surpass even the best model trained on the original dataset, which exhibits lower averaged performance metrics across all

weighted average metrics.

As expected, models are particularly well-suited for detecting the attack classes R2L and U2R. In the original NSL-KDD dataset, these classes suffer from a significant imbalance between their representation in the training and test splits, which previously hindered reliable detection.

Table 6.5: Results for models trained on the *balanced dataset* 4.3.

	First Run	Second Run	Third Run	Average
Accuracy	0,8456	0,8490	0,8492	0,8479
F1	0,8544	0,8550	0,8590	0,8561
Precision_score	0,8695	0,8665	0,8748	0,8703
Recall_score	0,8456	0,8490	0,8492	0,8479

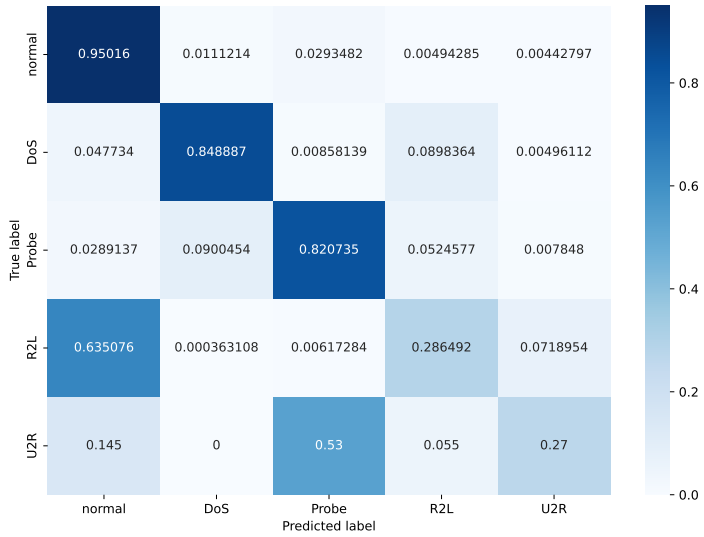

Figure 6.6: Confusion matrix: Test results of the *All* model, which was trained on all classes.

6.3 Experiment 2: Comparison of Computational Power

To evaluate the performance of models trained via the AE-RL framework we used various configurations. As in the original code, the Adam optimizer was used as the optimization method.

One hyperparameter for training is the learning rate (LR). Therefore, several values were tested: 0.00025, 0.0005, 0.00075, 0.001, 0.0015, and 0.0025. The best results were achieved with a learning rate of 0.001 is mentioned in more detail in Section 5.3. Slight

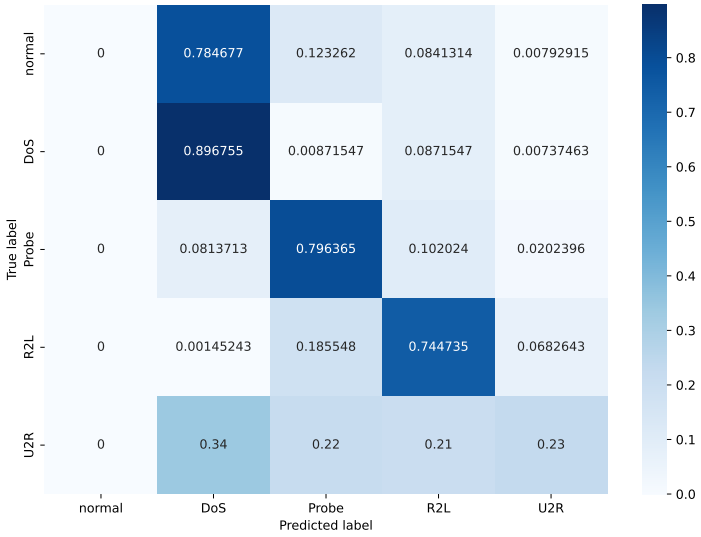

Figure 6.7: Confusion matrix: Test results of the *Attacks* model, which was trained on the attack classes `DoS`, `Probe`, `R2L`, and `U2R`.

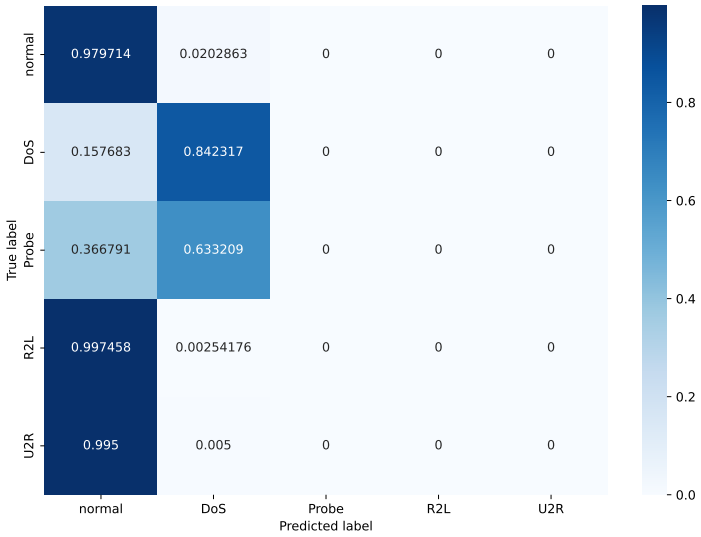

Figure 6.8: Confusion matrix: Test results of the model trained on the classes "Normal" and "DoS".

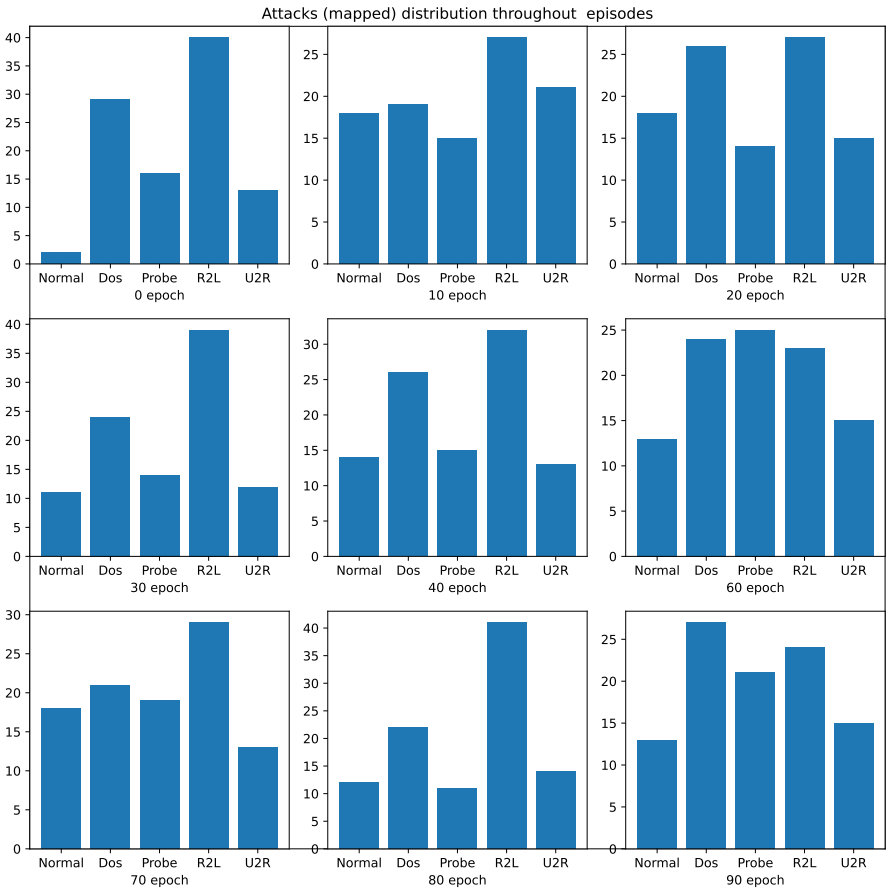

Figure 6.9: Overview of the distribution of attack types presented to the ***All*** model during training.

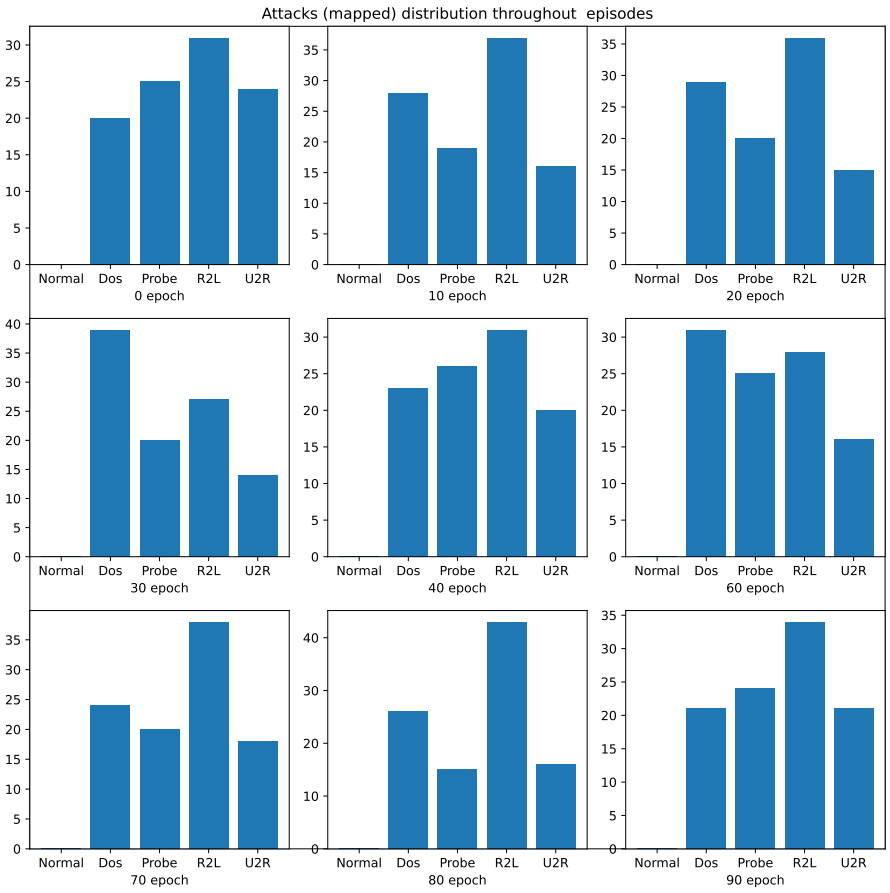

Figure 6.10: Overview of the distribution of attack types presented to the ***Attack*** model during training.

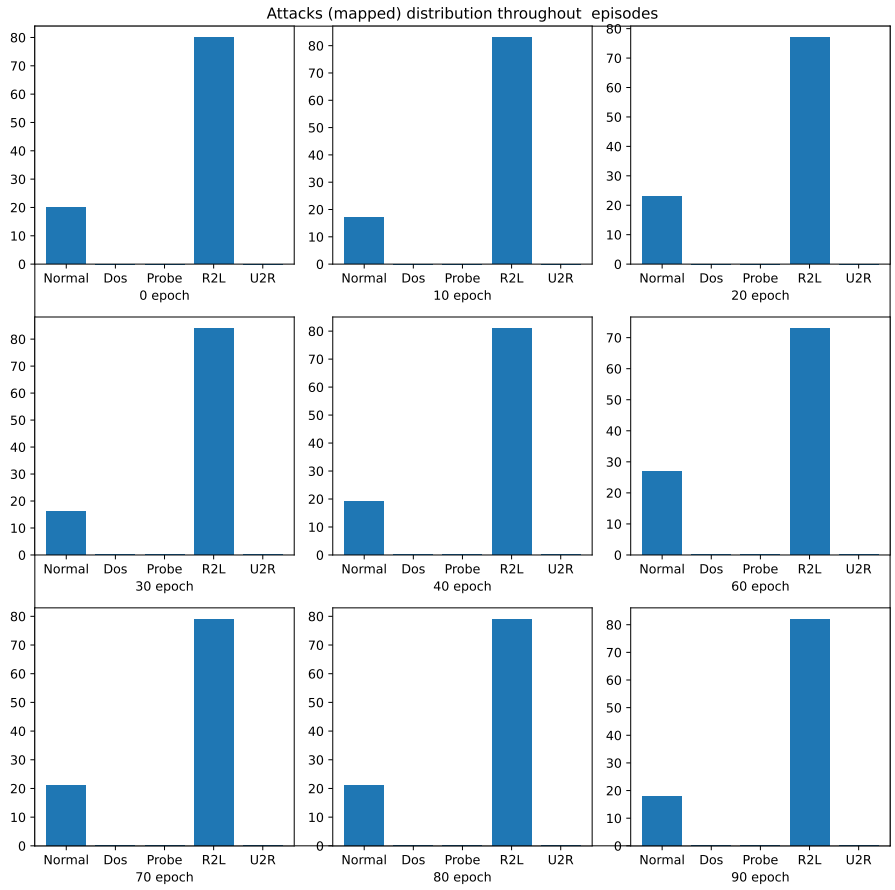

Figure 6.11: Overview of the distribution of attack types presented to the ***Normal & U2R*** model during training.

deviations around this value (±0.00025) led to similar results, while larger deviations increasingly degraded the model's performance.

Another important aspect was the architecture of the hidden layers in the models for the attacker and defender. Therefore three different configuration approaches were investigated:

1. Attacker model with more hidden layers than the defender.
2. Defender model with more hidden layers than the attacker.
3. Attacker and defender with the same number of hidden layers.

To improve the robustness of the results, each experiment was conducted three times per configuration. This allowed the calculation of an average value to minimize potential statistical outliers and enable a more stable evaluation of model performance.

1. Attacker model with more computational power:
The experiments showed that the best average performance was achieved with the combination of an attacker with 5 hidden layers (HL) and a defender with 3 HL at a learning rate of 0.001.

Closely following this was the configuration with Attacker = 5 HL, Defender = 3 HL, followed by Attacker = 4 HL, Defender = 3 HL, and Attacker = 6 HL, Defender = 3 HL. The worst results were achieved with Attacker = 8 HL against Defender = 3 HL. It can be observed that as the number of *hidden layers in the attacker model* increases, the model's performance gradually decreases.

- *Attacker = 4 HL, Defender = 3 HL → Accuracy: 0.8046, F1: 0.7991, Precision: 0.8051*
- ***Attacker = 5 HL, Defender = 3 HL → Accuracy: 0.8130, F1: 0.8019, Precision: 0.8050***
- *Attacker = 6 HL, Defender = 3 HL → Accuracy: 0.8005, F1: 0.7921, Precision: 0.7955*
- *Attacker = 8 HL, Defender = 3 HL → Accuracy: 0.7979, F1: 0.7862, Precision: 0.7890*

Examining the *distribution of selected attacks* for the two models *Attacker = 8 HL, Defender = 3 HL* and *Attacker = 4 HL, Defender = 3 HL* (see C.2), a remarkable difference becomes apparent. [2] While both models start with a *similar attack distribution* in *epoch 0*, the model with *Attacker = 4 HL* evolves over time into a *more balanced distribution* of selected attacks.

This observation is based on the analysis of *nine example epochs* (0, 10, 20, 30, 40, 60, 70, 80, and 90), where the progression of the attack strategy differs over the course of training. The attacker with *4 hidden layers* shows a *more differentiated selection of attacks*, while the model with *8 hidden layers* appears to remain more focused on certain types of attacks.

It seems that *too many hidden layers* cannot be optimally trained with that small training dataset. This leads to *reduced differentiation capability* and explains why the *best performing models* according to our metrics used a *configuration with Attacker = 5 HL and Defender = 3 HL*. This architecture seems to represent a *favourable compromise between model complexity and effective use of training data* for the NSL-KDD dataset.

2. Defender model with more computational power:
The best results were achieved with the configuration *Attacker = 1 HL, Defender = 3 HL*:

[2] It is easy to omit this fact because the y-axis within the visualization of the attack distribution is not normalized and therefore not optimal.

- ***Attacker = 1 HL, Defender = 3 HL → Accuracy: 0.8088, F1: 0.8032, Precision: 0.8047***
- *Attacker = 3 HL, Defender = 4 HL → Accuracy: 0.7902, F1: 0.7754, Precision: 0.7805*
- *Attacker = 3 HL, Defender = 5 HL → Accuracy: 0.7897, F1: 0.7744, Precision: 0.7826*
- *Attacker = 3 HL, Defender = 6 HL → Accuracy: 0.7880, F1: 0.7814, Precision: 0.7866*
- *Attacker = 3 HL, Defender = 8 HL → Accuracy: 0.7781, F1: 0.7680, Precision: 0.7786*
- *Attacker = 5 HL, Defender = 15 HL → Accuracy: 0.7728, F1: 0.7796, Precision: 0.8003*

These results show a clear trend:
As the number of hidden layers in the defender model increases, the overall model performance decreases. A likely reason for this is the *increasing number of parameters*, which may lead to *overfitting* relative to the size of the given training dataset.

Although the performance decreases only *slightly* with each additional layer the downward trend remains consistent. This suggests that *increasing the complexity of the defender model does not necessarily lead to better results* but, on the contrary, may impair the effectiveness of classification.

3. Comparison of equally powerful models:
To investigate the impact of the number of *hidden layers (HL)* on model performance, experiments were conducted where the attacker and defender each had the same number of layers. Models with *1 HL* to *8 HL* were tested, and their results were compared in terms of *accuracy*, *F1-score*, and *precision*.

The specific values for each tested configuration are presented in the following overview:

- *1 HL*: Accuracy = 0.7775, F1-Score = 0.7696, Precision = 0.7794
- *2 HL*: Accuracy = 0.7931, F1-Score = 0.7837, Precision = 0.7901
- *3 HL*: Accuracy = 0.7941, F1-Score = 0.7847, Precision = 0.7909
- ***4 HL*: Accuracy = 0.8026, F1-Score = 0.7895, Precision = 0.7938**
- *5 HL*: Accuracy = 0.7733, F1-Score = 0.7652, Precision = 0.7745
- *8 HL*: Accuracy = 0.7799, F1-Score = 0.7753, Precision = 0.7857

It can be observed that model performance increases with the number of *hidden layers* up to *4 HL*, but then continuously decreases. This may indicate that a *moderate network complexity* is optimal for the given task, while a too high number of HL may lead to overfitting or reduce the model's efficiency.

An overview of the test results can be found in the public available Google Docs spreadsheet [13]. All of the here mentioned experiments with their respective configurations and results are documented in the spreadsheet. In the github repository [14] all the code and scripts are available to reproduce the results as well as the trained models.

6.4 Summary of Architecture Experiments

The experiments on the architecture of the hidden layers provide *three key insights*:

1. *For models with the same number of hidden layers for attacker and defender*, a clear *performance improvement was observed up to 4 HL*, followed by a continuous *performance decline* in deeper networks.

   - The best result was achieved with *4 HL* (*Accuracy: 0.8026, F1: 0.7895, Precision: 0.7938*), closely followed by *3 HL*.
   - Networks with *more than 4 HL* showed a *decline in model performance*, which could indicate possible *overfitting or suboptimal weight distributions*.
2. *When comparing models where the attacker has more hidden layers than the defender*, the **overall best performance** was observed with ***Attacker = 5 HL, Defender = 3 HL*** (*Accuracy: 0.8130, F1: 0.8019, Precision: 0.8050*), closely followed by *Attacker = 4 HL, Defender = 3 HL*.
   - These configurations likely allow the attacker to *better differentiate attack strategies*, while the defender can efficiently respond to the identified patterns.
3. *The best results for a stronger defender architecture* were achieved with *Defender = 4 HL, Attacker = 3 HL*.
   - However, it was observed that an *attacker with only 1 HL against a defender with 3 HL* even delivered better results (*Accuracy: 0.8088, F1: 0.8032, Precision: 0.8047*).
   - This suggests that a simpler attacker challenges the *defender more effectively*, while excessive network depth in the attacker model tends to lead to performance losses.

Conclusion

The results show that *a moderate number of hidden layers* (approximately *3-4 HL*) is optimal for *effective model performance*. While deeper networks in the attacker model initially bring improvements, performance continuously declines with overly complex structures. Additionally, a configuration with *a weaker attacker (1 HL) and a stronger defender (3 HL)* seems to perform better than expected. This observation suggests that *balanced model complexity is crucial for classification performance* and that an overly powerful attacker during training does not necessarily lead to better results.

6.5 Experiment 3: Multi-Attacker Setup

To explore more realistic adversarial scenarios and increase the complexity of the learning environment, we extended the AE-RL framework to support multiple attacker agents. In this setup, the defender agent is simultaneously challenged by **four attacker agents**, each of which independently selects a sample (either benign or malicious) per iteration on a specific class of the dataset.

In this experiment, we used the NSL-KDD dataset, which contains five classes: `Normal`, `DoS`, `Probe`, `R2L`, and `U2R`. That means we have four attackers, each of which is specialized in one of the four attack classes: `DoS`, `Probe`, `R2L`, and `U2R`. Every attacker agent is trained to select either a sample from its own attack class or a benign sample from the `Normal` class. The defender agent, on the other hand, is trained to classify all presented attack classes and the `Normal` class.

Motivation

The original AE-RL framework assumed a one-to-one interaction between attacker and defender. However, in realistic network environments, systems are often targeted by multiple, concurrent attack sources. By introducing multiple attacker agents, we aim to simulate this dynamic and evaluate the defender's capacity to generalize and prioritize across parallel threats.

Setup and Training Configuration

Each attacker agent operates with its own Q-learning policy and maintains an independent Replay Memory. In each iteration:

- Every attacker selects one sample from the dataset, based on its learned policy it is either a benign sample from the `Normal` class or a malicious sample from its own attack class.
- The defender receives the four selected samples and must classify each one correctly.

The defender continues to choose among the five NSL-KDD classes: `Normal`, `DoS`, `Probe`, `R2L`, and `U2R`. Rewards are assigned as follows:

- If the defender correctly classifies a sample, it receives a positive reward.
- If the classification is incorrect, the responsible attacker is rewarded.

Asymmetric Experience Replay

As described detailed in Section 5.7, this multi-agent extension introduces asymmetry in experience collection: the defender accumulates four experiences per iteration, whereas each attacker only accumulates one. To address this imbalance, we increased the defender's Replay Memory capacity and minibatch size accordingly to 4000 and 400, respectively.

Goal

This experiment investigates how well the defender agent can cope with simultaneous diverse attack strategies. It further explores whether the defender benefits from increased exposure per episode or whether it suffers from policy instability due to overlapping or conflicting attacker behaviors.

Results

We conducted three training runs with different random seeds to ensure the robustness of the results. Training was performed over 100 epochs, with each epoch consisting of 100 iterations. Attacker agents were configured with 5 hidden layers (HL), while the defender agent used 3 HL. The learning rate was set to 0.001, and the Adam optimizer was applied for training.

Table 6.12 summarizes the results of the multi-agent setup. Against the expectation of improved performance through increased exposure to attack patterns, the results

show a decline in all performance metrics compared to the single-agent setup. These results demonstrates that the defender agent does not profit from the increased number of attack agents. The average performance metrics are lower than those achieved in the single-agent setup.

Table 6.12: Results of the multi-agent setup

	First Run	Second Run	Third Run	Average
Accuracy	0,8087	0,7949	0,7765	0,7934
F1	0,7864	0,78	0,7588	0,7751
Precision_score	0,8098	0,7957	0,7786	0,7786
Recall_score	0,8087	0,7949	0,7765	0,7934

When comparing the f1-score per class only the attack class *DoS* shows a slight improvement in the multi-agent setup in two of the three runs. However, as the result is not consistent across all runs, it is not possible to conclude that the multi-agent setup leads to a better detection of the *DoS* class. The attack classes *Probe*, *R2L*, and *U2R* show a significant decline in the f1-score compared to the single-agent setup. Confusion matrix 6.14 illustrates the classification performance of the multi-agent setup.

For the sake of completeness, we also provide the distribution of attack types presented to the multi-agent setup during training in Figure 6.13. Within the appendix we provide a distribution of the attacks selected by each attacker agent during training (see C.1).

These results suggest that the defender agent struggles to maintain robust classification performance in the presence of multiple simultaneous attack strategies.

A possible explanation for the decline in performance lies in the interplay between the increased volume of attacker decisions and the internal dynamics of the AE-RL training procedure. While the defender receives more samples per episode, it is unclear whether this leads to more effective learning or simply to faster memory saturation.

In addition, the AE-RL framework's dynamic sampling mechanism introduces stochasticity that may lead to inconsistent exposure to critical training patterns. Since not all training samples are guaranteed to be seen, it is conceivable that some models failed to observe representative instances of certain classes—particularly in a multi-class setup with multiple attackers.

Furthermore, the relative training frequency between attacker and defender might not be optimally balanced in the multi-agent scenario. If the defender updates its policy either too frequently or too infrequently compared to the attackers, learning instabilities may occur. These factors, in combination, could explain the observed drop in performance compared to the single-attacker setup.

6.6 Experiment 4: Boruta-Based Feature Selection

To evaluate the impact of feature selection on model performance, we conducted a series of experiments using the features identified by the Boruta algorithm, as described

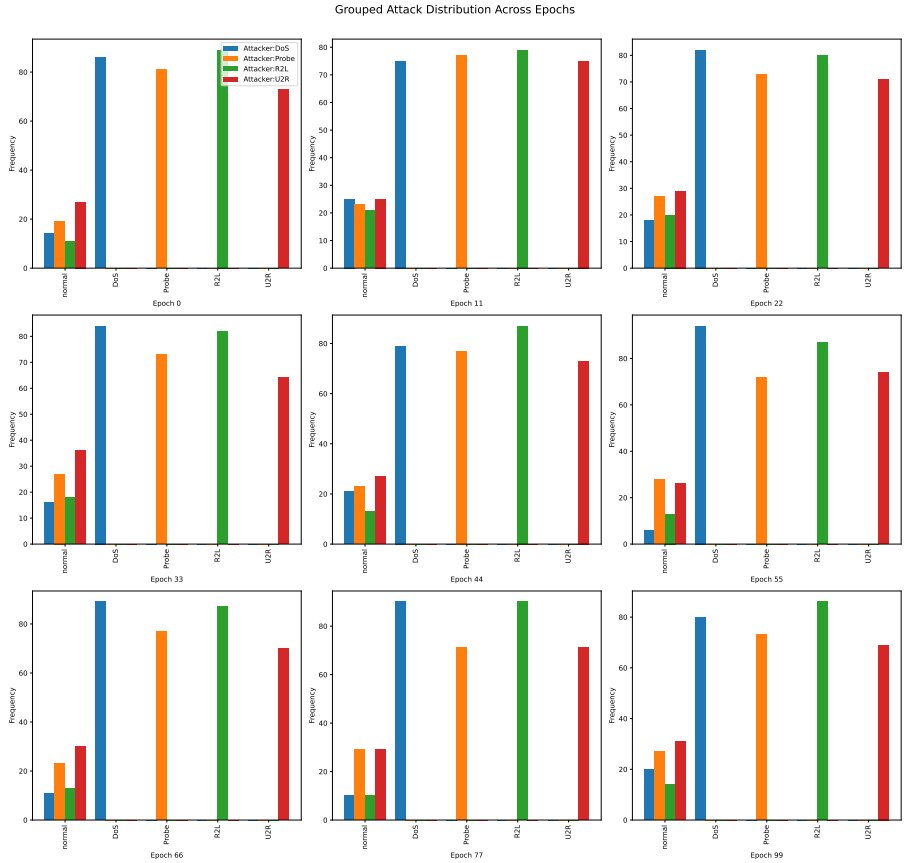

Figure 6.13: Overview of the distribution of attack types presented to the multi-agent setup during training. The distribution is shown for each attacker agent, which is specialized in one of the four attack classes: DoS, Probe, R2L, and U2R. The barplots show the number of samples selected by each attacker agent during training.

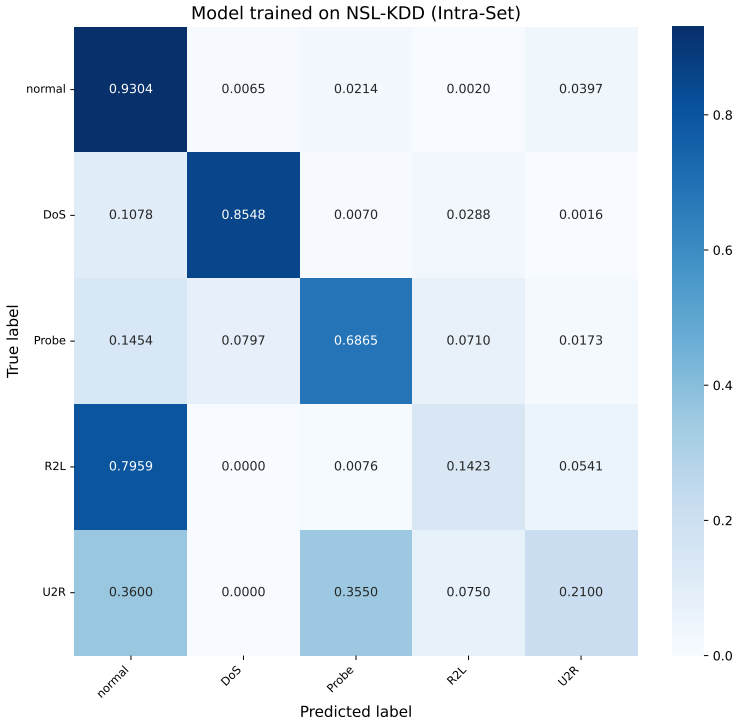

Figure 6.14: Confusion matrix: Test results of the model which was trained against multiple attackers.

in Section 5.4.

Two distinct feature sets were considered:

- **Core Features:** A compact subset of 24 features that were consistently marked as relevant across all dataset variants, including stratified, balanced, and attack-specific Boruta runs.
- **General Features:** A larger set of 42 features that were confirmed across the four general dataset variants (random, stratified, balanced, manually filtered).

Each setup was evaluated using the AE-RL framework with the configuration: `Attacker = 5 HL`, `Defender = 3 HL`, `learning rate = 0.001`, and 100 episodes. The NSL-KDD dataset was used in multi-class classification mode. Each experiment was repeated three times to account for the stochastic nature of both training and dynamic sample selection.

Results

The tables 6.15 and 6.16 summarize the average classification performance for both feature sets.

Table 6.15: Model performance using **Core Features** (24 features).

Metric	First Run	Second Run	Third Run	Average
Accuracy	0.7552	0.7509	0.7629	0.7563
F1	0.7407	0.7398	0.7501	0.7435
Precision	0.7441	0.7398	0.7466	0.7435
Recall	0.7552	0.7509	0.7629	0.7563

Table 6.16: Model performance using **General Features** (42 features).

Metric	First Run	Second Run	Third Run	Average
Accuracy	0.7835	0.8135	0.7897	0.7956
F1	0.7722	0.8030	0.7821	0.7858
Precision	0.7904	0.8067	0.7862	0.7944
Recall	0.7835	0.8136	0.7897	0.7956

Discussion

The models trained on the **General Features** achieved higher average scores across all metrics compared to those trained on the **Core Features**. This indicates that including a broader—but still filtered—set of relevant features can lead to better performance, likely due to increased representational capacity and diversity in feature space.

However, it is important to note that both setups showed visible variance across runs, especially in F1-score. This suggests that the stochastic nature of the AE-RL sampling

strategy may still dominate performance, regardless of feature subset size. Even with a reduced and more targeted feature set, inconsistencies in sample exposure can lead to diverging results between training runs.

Consequently, while Boruta-based feature selection is a valuable preprocessing step for improving model interpretability and potentially reducing overfitting, its full benefit can only be realized when paired with more consistent sampling and experience replay strategies—as discussed in Chapter 7.

The complete list of confirmed features for the *Core Feature Set* and *General Feature Set* used in this experiment is provided in Appendix E.1.

6.7 Experiment 5: CIC-IDS – Generalization Capabilities

Following the approach of Verkerken et al.[21], we conducted a series of experiments to evaluate the generalization performance of the AE-RL framework. Specifically, we trained models on the CIC-IDS-2017 dataset and evaluated them on the CIC-IDS-2018 dataset. For a detailed overview of the training configuration and data preparation, refer to Section5.8.

The training setup mirrored that of previous experiments, using the Adam optimizer with a learning rate of 0.001. Each model was trained for 100 episodes, with 100 iterations per episode. In contrast to earlier experiments, all evaluation metrics in this setting were calculated *without* applying class-weighted averaging, as the classification task was binary.

A notable observation emerged during training on the DDOS class: while three models achieved high accuracy in the intra-dataset evaluation, their performance varied drastically on the inter-dataset evaluation. Two models experienced moderate performance drops of up to 16 percentage points in F1-score, while the third exhibited a dramatic decline—dropping to an F1-score of just 41%, a decrease of 47 percentage points compared to its intra-dataset result.

In light of this high variance we trained two additional models for the DDOS class. These achieved inter-dataset F1-scores of 93% and 74%, respectively. All five models showed consistently high performance in the intra-dataset evaluation, with F1-scores of 97% or higher. However, the large variation in inter-dataset results indicates a lack of consistency in generalization.

This suggests that while the AE-RL framework is capable of achieving strong performance on known data distributions, it still lacks robustness in generalizing to unseen traffic patterns across datasets. A likely explanation lies in the dynamic sampling strategy introduced by the AE-RL framework [5]. Although this mechanism is one of its core strengths, it may also be the source of instability in model behavior.

As discussed in Section 3.7, the current implementation does not guarantee that the model is exposed to the entire training dataset. Instead, training is limited to a random dynamically selected subset of data, which may result in models failing to learn all relevant patterns present in the source dataset.

This leads to the observed inconsistency: some models generalize well, while others fail to do so—sometimes dramatically. This issue is not limited to inter-dataset evalua-

tion; similar behavior was observed in intra-dataset evaluations, particularly on the NSL-KDD dataset.

In summary, the AE-RL framework shows promising potential for generalization but currently lacks consistency across repeated training runs. Future improvements should aim to refine the sampling strategy to ensure broader and more representative data exposure during training.

Comparison with Verkerken et al.

Although Verkerken et al. provide averaged F1-scores across all attack types, we refrain from such aggregation due to high performance variance observed in our reinforcement learning setup. Instead, we report representative results for a single attack class (DDoS), as this best illustrates both the strengths and limitations of our model.

Evaluation on CIC-IDS-2017 and CIC-IDS-2018 (DDOS)

To evaluate the generalization capability of the AE-RL framework, we followed the experimental setup proposed by Verkerken et al. [21] and trained models on one dataset while evaluating on another. This subsection focuses on the DDOS class, which is present in both CIC-IDS-2017 and CSE-CIC-IDS-2018 and provides a suitable basis for comparison.

We trained three models on each dataset for intra- and inter-dataset evaluation using the same configuration: 5 hidden layers for the attacker, 3 for the defender, a learning rate of 0.001, and 100 training episodes. Evaluation results are reported using four standard classification metrics: Accuracy, F1 score, Precision, and Recall. All metrics are computed using scikit-learn's default binary classification evaluation without class weighting (i.e., `average=None`).

Intra-dataset Evaluation.
Models trained and validated on the same dataset (intra-set evaluation) yielded strong and consistent performance. For CIC-IDS-2017, the average F1 score was 0.9806, and for CIC-IDS-2018, even higher at 0.9984 (see Table 6.18 and Table 6.17). These results indicate that the model can effectively learn to detect DDOS traffic patterns within the same dataset distribution.

Table 6.17: Intra-dataset evaluation: Training and testing on CIC-IDS-2017 (DDOS). Configuration: 5 hidden layers for attacker, 3 for defender, learning rate = 0.001.

Metric	First Run	Second Run	Third Run	Average
Accuracy	0.9702	0.9830	0.9755	0.9762
F1	0.9754	0.9861	0.9803	0.9806
Precision	0.9836	0.9808	0.9636	0.9760
Recall	0.9674	0.9916	0.9976	0.9855

Inter-dataset Evaluation.
In contrast, inter-dataset performance revealed substantial variance. When trained on

Table 6.18: Intra-dataset evaluation: Training and testing on CIC-IDS-2018 (DDOS). Configuration: 5 hidden layers for attacker, 3 for defender, learning rate = 0.001.

Metric	First Run	Second Run	Third Run	Average
Accuracy	0.9977	0.9976	0.9968	0.9974
F1	0.9986	0.9986	0.9981	0.9984
Precision	0.9981	0.9992	0.9984	0.9986
Recall	0.9991	0.9992	0.9977	0.9987

CIC-IDS-2017 and tested on CIC-IDS-2018, F1 scores ranged from 0.4135 to 0.8278 across runs, averaging 0.6830 (Table 6.19). This shows some generalization ability, but also highlights instability across different runs.

The reverse direction—training on CIC-IDS-2018 and testing on CIC-IDS-2017—performed worse, with average F1 of only 0.2879 and recall as low as 0.1926 (Table 6.20). While intra-dataset results were near-perfect in this setup, inter-dataset results indicate that the model fails to generalize reliably in this direction.

Table 6.19: Inter-dataset evaluation: Training on CIC-IDS-2017, testing on CIC-IDS-2018 (DDOS). Configuration: 5 hidden layers for attacker, 3 for defender, learning rate = 0.001.

Metric	First Run	Second Run	Third Run	Average
Accuracy	0.7637	0.7938	0.4972	0.6849
F1	0.8076	0.8278	0.4135	0.6830
Precision	0.8707	0.9199	0.8920	0.8942
Recall	0.7531	0.7250	0.2692	0.5824

Table 6.20: Inter-dataset evaluation: Training on CIC-IDS-2018, testing on CIC-IDS-2017 (DDOS). Configuration: 5 hidden layers for attacker, 3 for defender, learning rate = 0.001.

Metric	First Run	Second Run	Third Run	Average
Accuracy	0.5104	0.4337	0.4774	0.4738
F1	0.4714	0.1404	0.2519	0.2879
Precision	0.6915	0.9569	0.2519	0.6334
Recall	0.3580	0.0758	0.1441	0.1926

Observations and Discussion.
These results suggest that while the AE-RL framework performs strongly in intra-dataset evaluations, it struggles with generalization across datasets, particularly when trained on CIC-IDS-2018 and tested on CIC-IDS-2017. The sharp drop in recall in some runs (as low as 7.5%) implies that certain models fail to detect DDOS traffic almost entirely under domain shift.

A likely explanation is the dynamic and selective sampling behavior inherent to the AE-RL setup: during training, not all samples are seen by the agent. Depending on explo-

ration patterns and the attack distribution, the model may overfit to specific characteristics of the training data. This may explain why some runs generalize well while others do not, despite using the same configuration.

Conclusion.
Overall, the inter-dataset results highlight a key limitation of the AE-RL approach: its sensitivity to sampling and initialization makes generalization inconsistent. While intra-dataset detection is excellent, further work is needed to stabilize cross-domain performance. Possible improvements include better coverage of the training space, curriculum learning, or using regularization techniques that promote robustness.

6.8 Summary of Experiments

This chapter presented various experiments to analyze the impact of training data and model architecture on classification performance. The data was divided into different subsets, and models were trained with various configurations. The key findings can be summarized as follows:

Impact of Training Data

The experiments demonstrated that the choice of training data significantly influences model performance. Notably, the ***Attacks model***, trained only on attack classes *DoS, Probe, R2L*, and *U2R*, recognized the *R2L* class significantly better than the ***All model***, which was trained on all classes. The results suggest that a targeted selection of training data can be advantageous for better classification of specific attack classes.

The experiments with **two-class models** (e.g., *Normal & DoS*) showed that these models achieved *high detection performance for the trained classes* but generalized worse than models trained on multiple attack classes. Another approach was training on an **equally distributed set of normal and anomalous instances**, which, however, did not lead to performance improvements. Instead, reducing the number of normal instances in training proved to be detrimental, as important patterns were lost.

The experiments with the **balanced dataset** in case of the NSL-KDD dataset showed that the model performance improved significantly, especially for the under-represented classes *R2L* and *U2R*. This indicates that a balanced dataset can help the defender model to learn more representative decision boundaries and improve classification performance.

Impact of Model Architecture: NSL-KDD

The model architecture, particularly the number of *hidden layers (HL)* in the attacker and defender networks, played a crucial role in classification performance. The experiments yielded three key insights:

1. Models with *the same number of hidden layers* for attacker and defender showed **optimal performance at 4 HL**. Beyond this point, model performance continuously declined, likely due to *overfitting* or inefficient use of network complexity.

2. When the *attacker had more hidden layers than the defender*, the combination ***Attacker = 5 HL, Defender = 3 HL*** performed best. This suggests that a moderately complex attacker can be advantageous for differentiated attack selection depending on the used dataset.
3. Conversely, a *stronger defender architecture* with ***Defender = 3 HL, Attacker = 1 HL*** also delivered good results. This indicates that a less complex attacker model can generate more challenging attack patterns without relying on overly specific strategies.

Conclusion

The results of these experiments demonstrate that both *training data* and *network architecture* play a significant role in classification performance. A targeted selection of attack classes can improve the detection of specific attacks, while a moderate number of *hidden layers* creates an effective balance between model complexity and generalization capability.

The **best overall model** was achieved with the architecture ***Attacker = 5 HL, Defender = 3 HL*** at a learning rate of 0.001 using the Adam optimizer. This model achieved the best balance between precision, F1-score, and robust classification capability.

Impact of Multi-Agent Training

In Experiment 6.5, we extended the AE-RL framework to support multiple concurrent attacker agents. Contrary to expectations, the defender agent did not benefit from the increased exposure to diverse attack patterns. Performance metrics declined across all runs compared to the single-attacker setup, with F1-scores averaging 0.7751. Only the DoS class showed minor improvements in some runs, but these were not consistent.

These findings suggest that the defender model may struggle when simultaneously exposed to multiple attack policies, likely due to conflicting signals and limited model capacity. Moreover, the increased number of samples per iteration led to faster Replay Memory turnover, potentially contributing to instability in learning. Future work should explore mechanisms such as prioritized replay, attack scheduling, or separate classification heads to stabilize training in multi-agent settings.

Generalization to Unseen Data

Experiment 6.7 evaluated the generalization capability of the AE-RL framework using CIC-IDS-2017 and CSE-CIC-IDS-2018 datasets in an inter-dataset evaluation. While intra-dataset results were strong (F1-scores up to 0.998), inter-dataset results varied greatly—ranging from 0.2879 to 0.8278 in F1-score—even under identical configurations.

These inconsistencies highlight a key limitation of the current AE-RL setup: while powerful within a known distribution, it lacks robustness under domain shifts. This can be attributed to the selective sampling behavior of the attacker agent, which may prevent the defender from seeing a representative portion of the training data. Models trained in this way may overfit to specific characteristics of the training set and fail to generalize to new data distributions.

To improve generalization, future versions of the AE-RL framework could benefit from:

- **Curriculum learning** strategies to gradually expose the model to more diverse or difficult attack types,
- **Coverage guarantees** that ensure every sample is seen during training, or
- **Regularization techniques** that improve robustness to domain shift.

Final Remarks

Across all experiments, the AE-RL framework demonstrated a high degree of flexibility in adapting to various training scenarios and architectures. However, the results also revealed its sensitivity to data imbalance, architectural configurations, and stochastic training dynamics. While the model achieved competitive performance on intra-dataset evaluations and responded well to dataset balancing and architectural tuning, its generalization capability across datasets remained inconsistent.

These findings highlight the importance of both controlled experimental design and robust training strategies in reinforcement learning–based intrusion detection. In particular, consistent generalization across datasets remains an open challenge and serves as a key motivation for future work. The insights gained from these experiments serve as a foundation for refining the AE-RL framework toward more stable and transferable detection performance.

7

Future Work

7.1 Challenges and Limitations

The inter-dataset evaluation revealed considerable variability in model performance depending on the attack class and the specific training run. While intra-dataset results were consistently strong—frequently achieving F1-scores above 97%—inter-dataset scores varied dramatically, ranging from 41% to 93% for the same configuration on `(D)DoS`.

A likely explanation lies in the dynamic sampling strategy of the AE-RL framework. While the interactive selection of training samples is one of the framework's core strengths, it can also lead to instability and incomplete coverage: due to the evolving Q-values, certain classes or regions of the dataset may be under-represented during training. This results in high variance across repeated runs and inconsistent generalization to unseen data.

7.2 Future Directions

To enhance the robustness, interpretability, and applicability of the AE-RL framework, we propose the following avenues for future work:

1. Visualization and Coverage Analysis

Introduce systematic tools to monitor which samples and classes were actually used during training. This could include logging the indices of selected samples and calculating per-class selection frequencies. Such visualizations would help to understand whether coverage imbalances correlate with poor generalization and allow researchers to intervene accordingly.

2. Replay Coverage Control

Ensure that the defender agent is exposed to all training samples at least once, or at least in proportion to class frequencies. This can be realized through stratified sampling, static warm-up phases, or coverage-aware experience replay buffers.

3. Randomized Attack Sampling as a Baseline

Replace the attacker agent with a simple randomized attack strategy that selects samples proportionally to class frequencies. This setup can serve both as a strong baseline and as a mechanism to enforce more balanced exposure to rare attack types—especially during early training stages.

4. Curriculum Learning

Incorporate a staged training setup that introduces attack types gradually. Models could first be trained on more frequent attacks before being exposed to rare or complex ones (e.g., U2R or `Infiltration`). Such a curriculum could improve learning stability and class coverage.

5. Adaptive Model Architectures

Explore dynamic architectures that adjust the model complexity (e.g., number of hidden layers or neurons) based on the dataset characteristics. Such a mechanism could help tailor the model to different data regimes without manual tuning.

6. Uncertainty-Aware Classification

Introduce an additional output class for "uncertain" or "unfamiliar" samples. Such predictions could be flagged for human review in a real-world intrusion detection system, helping reduce false positives while enabling targeted data collection for future labeling.

7. Hierarchical Defenders

Evaluate a hierarchical classification approach: one defender agent first performs a binary decision (malicious vs. benign), and only if malicious, a second defender performs fine-grained attack-type classification. This could reduce misclassification of benign traffic and isolate complex decisions to relevant cases only.

8. Reward Function Refinement

Investigate modified or multi-stage reward schemes that better align with classification objectives, especially for imbalanced datasets or under domain shift. Alternative rewards may guide the defender toward more robust class-discriminative decision boundaries.

9. Regularization and Robustness Strategies

Incorporate regularization techniques such as dropout, early stopping, or data augmentation to reduce variance across repeated runs. This could be especially valuable in inter-dataset settings where domain gaps are known to impair generalization.

7.3 Outlook

While the AE-RL framework shows considerable potential in simulating adversarial learning dynamics, its success currently depends heavily on stable training behavior and balanced sample selection. The proposed improvements target precisely these weaknesses by increasing transparency, improving sample coverage, and reducing model variance. Addressing them will be crucial to deploying such frameworks in real-world network intrusion detection systems.

8

Conclusion

This thesis critically examined and extended the AE-RL framework originally proposed by Caminero et al. [5], with the aim of analysing its performance in various training configurations and its ability to generalize to new datasets. Rather than proposing a completely new architecture, the primary contribution of this work lies in a detailed empirical analysis of the framework, the identification of architectural limitations, and a thorough evaluation across multiple datasets.

We began by reproducing and scrutinizing the original training setup, revealing inconsistencies between the implementation and the claims made in the associated publication 3.7. In particular, we showed that the assumption that all training data is seen by the defender during each episode does not hold in practice. This insight has important implications for the interpretation of the model's performance and was highlighted as a key area for future improvement.

The AE-RL framework was extended to support configurable training scenarios, including binary classification in addition to multi-class, as well as the One-vs-All strategy. This enhancement enabled targeted experiments that shed light on the influence of data distribution, attack class frequency, and network complexity on the model's classification performance.

Architecture-wise, we empirically investigated various attacker-defender configurations and found that a moderate number of hidden layers—typically 3 to 5—yielded the best trade-off between accuracy and stability. Overly deep networks tended to overfit or underperform, especially when trained on limited or imbalanced datasets.

To assess generalization capabilities, we applied the AE-RL framework to two real-world intrusion detection datasets: CIC-IDS-2017 and CSE-CIC-IDS-2018. Our inter-dataset evaluation revealed high variability across runs, suggesting that the model's generalization depends heavily on sampling behavior and initialization. This underlines the need for improved training procedures and more robust evaluation practices.

Overall, this work contributes to the AE-RL research field by uncovering important implementation-specific issues, extending the framework for broader experimentation, and highlighting its strengths and limitations in practical deployment scenarios. The results presented here provide a foundation for further improvements and serve as a baseline for future work in reinforcement learning-based intrusion detection.

9

Code Repository and Reproducibility

All experiments were conducted locally on a machine equipped with an AMD Ryzen 7 5800X CPU, 16 GB of RAM, and an NVIDIA GeForce RTX 5080 GPU (16 GB VRAM), running Ubuntu 24.04.2 LTS. The software environment was provided via the official NVIDIA TensorFlow Docker container: `nvcr.io/nvidia/tensorflow:25.02-tf2-py3`.

The developed AE-RL framework extension build upon the codebase presented in the original paper *Adversarial Environment Reinforcement Learning for Network Intrusion Detection* [5], which is publicly available at:
https://github.com/gcamfer/Anomaly-ReactionRL/blob/master/Notebooks/AE_RL_NSL_KDD.ipynb

All extensions, modifications, and training pipelines developed in this thesis are available in the following GitHub repository:
https://github.com/vlad961/AE-RL-Python [14].

An overview of most conducted experiments, configurations, and performance metrics has been documented in a public Google Spreadsheet:
https://docs.google.com/spreadsheets/d/1Uak1kR54UffO9Q2jjN1uvLxOFaKkG0_76U_Fm_ayMg8/edit?usp=sharing

All pre-trained models and evaluation results are located in the repository under:
`root/models/trained-models`

All experiments were executed locally, and results can be fully reproduced using the provided training scripts and configuration files.

Author's Note on Language Assistance

Parts of this thesis were linguistically revised and refined with the help of the language model `ChatGPT` by OpenAI. The tool was used exclusively for improving clarity, grammar, and stylistic expression. All scientific contributions, reasoning, and conclusions presented herein are my own and were — to the best of my knowledge and intent — critically examined and validated for correctness and academic integrity.

Bibliography

[1] Bhuyan, M. H., Bhattacharyya, D. K., and Kalita, J. K. Network Anomaly Detection: Methods, Systems and Tools. In: *IEEE Communications Surveys & Tutorials* 16(1):303–336, 2014. DOI: 10.1109/SURV.2013.052213.00046.

[2] Botacin, M. GPThreats-3: Is Automatic Malware Generation a Threat? In: *2023 IEEE Security and Privacy Workshops (SPW)*. 2023, pp. 238–254. DOI: 10.1109/SPW59333.2023.00027.

[3] Bundesamt für Bevölkerungsschutz und Katastrophenhilfe (BBK) *Cybergefahren für kritische Infrastrukturen [Cyber Threats for Critical Infrastructures]*. https://www.bbk.bund.de/DE/Themen/Kritische-Infrastrukturen/KRITIS-Gefahrenlagen/Cybergefahren/cybergefahren_node.html. Accessed on April 28, 2025. 2024.

[4] Camfer, G. and López-Martín *Anomaly Reaction Reinforcement Learning for Intrusion Detection Github Repository*. Accessed on April 8, 2025. 2025. URL: https://github.com/gcamfer/Anomaly-ReactionRL.

[5] Caminero, G., Lopez-Martin, M., and Carro, B. Adversarial environment reinforcement learning algorithm for intrusion detection. In: *Computer Networks* 159:96–109, 2019. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2019.05.013. URL: https://www.sciencedirect.com/science/article/pii/S1389128618311216.

[6] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. SMOTE: Synthetic Minority Over-sampling Technique. In: *Journal of Artificial Intelligence Research* 16:321–357, June 2002. ISSN: 1076-9757. DOI: 10.1613/jair.953. URL: http://dx.doi.org/10.1613/jair.953.

[7] da Costa, K. A., Papa, J. P., Lisboa, C. O., Munoz, R., and de Albuquerque, V. H. C. Internet of Things: A survey on machine learning-based intrusion detection approaches. In: *Computer Networks* 151:147–157, 2019. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2019.01.023. URL: https://www.sciencedirect.com/science/article/pii/S1389128618308739.

[8] Deutscher Bundestag *Cyberangriffe auf akademische und wissenschaftliche Einrichtungen: Antwort auf die Kleine Anfrage der CDU/CSU-Fraktion [Reply of the Federal Government to the Minor Interpellation of the CDU/CSU Parliamentary Group: Cyberattacks on Academic and Research Institutions]*. Drucksache 20/11830, accessed on April 28, 2025. 2024. URL: https://www.bundestag.de/presse/hib/kurzmeldungen-1013548.

[9] He, H., Bai, Y., Garcia, E. A., and Li, S. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In: *2008 IEEE International Joint Conference on Neu-*

ral Networks (IEEE World Congress on Computational Intelligence). 2008, pp. 1322–1328. DOI: 10.1109/IJCNN.2008.4633969.

[10] Kursa, M. B. and Rudnicki, W. R. Feature Selection with the Boruta Package. In: *Journal of Statistical Software* 36(11):1–13, 2010. DOI: 10.18637/jss.v036.i11. URL: https://www.jstatsoft.org/index.php/jss/article/view/v036i11.

[11] Liu, L., Engelen, G., Lynar, T., Essam, D., and Joosen, W. Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018. In: *2022 IEEE Conference on Communications and Network Security (CNS)*. 2022, pp. 254–262. DOI: 10.1109/CNS56114.2022.9947235.

[12] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12:2825–2830, 2011.

[13] Sehtman, V. *Performance Comparison of models trained in the AE-RL Framework*. https://docs.google.com/spreadsheets/d/1Uak1kR54UffO9Q2jjN1uvLxOFaKkG0_76U_Fm_ayMg8. Public spreadsheet created during my master's thesis. 2025.

[14] Sehtman, V. *Performance Comparison of models trained in the AE-RL Framework and extension of the AE-RL framework*. https://github.com/vlad961/AE-RL-Python. Public Gitlab repository created during my master's thesis. 2025.

[15] Sharafaldin, I., Lashkari, A. H., and Ghorbani, A. A. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In: *International Conference on Information Systems Security and Privacy*. 2018. URL: https://api.semanticscholar.org/CorpusID:4707749.

[16] Stolfo, S., Fan, W., Lee, W., Prodromidis, A., and Chan, P. Cost-based modeling for fraud and intrusion detection: results from the JAM project. In: *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*. Vol. 2. 2000, 130–144 vol.2. DOI: 10.1109/DISCEX.2000.821515.

[17] Sutton, R. and Barto, A. Reinforcement Learning: An Introduction. In: *IEEE Transactions on Neural Networks* 9(5):1054–1054, 1998. DOI: 10.1109/TNN.1998.712192.

[18] Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. A detailed analysis of the KDD CUP 99 data set. In: *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*. 2009, pp. 1–6. DOI: 10.1109/CISDA.2009.5356528.

[19] Universitätsklinikum Düsseldorf *IT-Ausfall an der Uniklinik Düsseldorf: Cyberangriff bestätigt – Sicherheitslücke in verbreiteter Software ermöglichte Zugang [IT Outage at the University Hospital Düsseldorf: Cyberattack Confirmed - Security Vulnerability in Common Software Allowed Access]*. https://www.uniklinik-duesseldorf.de/ueber-uns/pressemitteilungen/detail/it-ausfall-an-der-uniklinik-duesseldorf. Accessed on April 20, 2025. 2020.

[20] Verkerken, M. *IDS Dataset Cleaning Gitlab Repository*. Accessed on March 8, 2025. 2025. URL: https://gitlab.ilabt.imec.be/mverkerk/ids-dataset-cleaning.

[21] Verkerken, M., D'hooge, L., Wauters, T., Volckaert, B., and De Turck, F. Towards Model Generalization for Intrusion Detection: Unsupervised Machine Learning Techniques. In: *Journal of Network and Systems Management* 30(1):12, Oct. 2021. DOI: 10.1007/s10922-021-09615-7. URL: https://doi.org/10.1007/s10922-021-09615-7.

A

Overview of the NSL-KDD Dataset

A.1 General Information

Visual representation of the distribution of considered attack types and attacks in the NSL-KDD dataset can be found in 4.7 and 4.8.

A.2 Attack Types and Distributions

Table A.1: Considered attack types in the *test dataset*.

Attack Type	Count	Rounded Share in %
normal	9.711	43,1 %
DoS	7.458	33,1 %
Probe	2.421	10,7 %
R2L	2.754	12,2 %
U2R	200	0,89 %

Table A.2: Considered *attack types* in the training dataset.

Attack Type	Number of Attacks	Share in %
normal	67.343	53,4582 %
DoS	45.927	36,4578 %
Probe	11.656	9,252 %
R2L	995	0,789 %
U2R	52	0,04 %

Table A.3: Overview of all attack types and their specific attacks.

Attack Type	Attack
Normal	normal
DoS	back, land, neptune, pod, smurf, teardrop, **mailbomb, apache2, processtable, udpstorm**
Probe	ipsweep, nmap, portsweep, satan, **mscan, saint**
R2L	ftp_write, guess_passwd, imap, multihop, phf, *spy*, *warezclient*, warezmaster, **sendmail, named, snmpgetattack, snmpguess, xlock, xsnoop, worm**
U2R	buffer_overflow, loadmodule, perl, rootkit, **httptunnel, ps, sqlattack, xterm**

Note: Bold attacks (17) are only present in the test dataset. Italicized attacks (2) are only present in the training dataset. The remaining attacks are present in both datasets. In total, there are 22 attacks in the training dataset (+1 normal label) and 37 attacks in the test dataset (+1 normal label).

A.3 Details of the Categorical Features

Here are the three categorical features, 'protocol_type', 'flag', and 'service', listed along with the possible values they can take in the NSL-KDD dataset.

Protocol_type has 3 possible values: `{'tcp', 'udp', 'icmp' }`

Flag has 11 possible values: `{'OTH', 'REJ', 'RSTO', 'RSTOS0', 'RSTR', 'S0', 'S1', 'S2', 'S3', 'SF', 'SH' }`

Service has 70 possible values: `{'aol', 'auth', 'bgp', 'courier', 'csnet_ns', 'ctf', 'daytime', 'discard', 'domain', 'domain_u', 'echo', 'eco_i', 'ecr_i', 'efs', 'exec', 'finger', 'ftp', 'ftp_data', 'gopher', 'harvest', 'hostnames', 'http', 'http_2784', 'http_443', 'http_8001', 'imap4', 'IRC', 'iso_tsap', 'klogin', 'kshell', 'ldap', 'link', 'login', 'mtp', 'name', 'netbios_dgm', 'netbios_ns', 'netbios_ssn', 'netstat', 'nnsp', 'nntp', 'ntp_u', 'other', 'pm_dump', 'pop_2', 'pop_3', 'printer', 'private', 'red_i', 'remote_job', 'rje', 'shell', 'smtp', 'sql_net', 'ssh', 'sunrpc', 'supdup', 'systat', 'telnet', 'tftp_u', 'tim_i', 'time', 'urh_i', 'urp_i', 'uucp', 'uucp_path', 'vmnet', 'whois', 'X11', 'Z39_50'}`

Table A.4: Considered *attacks* in the *training dataset*.

Attack Label	Number of Attacks (Total = 125.973)
normal	67.343
back	956
land	18
neptune	41.214
pod	201
smurf	2.646
teardrop	892
mailbomb	0
apache2	0
processtable	0
udpstorm	0
ipsweep	3.599
nmap	1493
portsweep	2.931
satan	3.633
mscan	0
saint	0
ftp_write	8
guess_passwd	53
imap	11
multihop	7
phf	4
spy	2
warezclient	890
warezmaster	20
sendmail	0
named	0
snmpgetattack	0
snmpguess	0
xlock	0
xsnoop	0
worm	0
buffer_overflow	30
loadmodule	9
perl	3
rootkit	10
httptunnel	0
ps	0
sqlattack	0
xterm	0

Table A.5: Considered *attacks* without actual attack samples in the training dataset.

Attack Label	Number of Attacks
mailbomb (DoS)	0
apache2 (DoS)	0
processtable (DoS)	0
udpstorm (DoS)	0
mscan (Probe)	0
saint (Probe)	0
sendmail (R2L)	0
named (R2L)	0
snmpgetattack (R2L)	0
snmpguess (R2L)	0
xlock (R2L)	0
xsnoop (R2L)	0
worm (R2L)	0
httptunnel (U2R)	0
ps (U2R)	0
sqlattack (U2R)	0
xterm (U2R)	0

Table A.6: Considered *attacks* in the *test dataset*.

Attack Label	Number of Attacks (Total = 22.544)
normal	9.711
back	359
land	7
neptune	4.657
pod	41
smurf	665
teardrop	12
mailbomb	293
apache2	737
processtable	685
udpstorm	2
ipsweep	141
nmap	73
portsweep	157
satan	735
mscan	996
saint	319
ftp_write	3
guess_passwd	1.231
imap	1
multihop	18
phf	2
spy	0
warezclient	0
warezmaster	944
sendmail	14
named	17
snmpgetattack	178
snmpguess	331
xlock	9
xsnoop	4
worm	2
buffer_overflow	20
loadmodule	2
perl	2
rootkit	13
httptunnel	133
ps	15
sqlattack	2
xterm	13

Table A.7: Overview of the ratios of the number of attacks per attack type in the *test dataset/training dataset*.

Attack Type	*Test Dataset/Training Dataset*	Ratio
Normal	9711/67343	0,1442
DoS	7458/43727	0,1706
Probe	2421/11656	0,2077
R2L	2754/995	2,7678
U2R	200/52	3,8462

B

Additional Resources and Complete Experimental Results

Due to the large number of experiments conducted, not all configurations and results are detailed in this thesis. The complete results can be accessed in the following repository:

`https://github.com/vlad961/AE-RL-Python/tree/main/models/trained-models`

For each trained model, the configuration parameters and the achieved results are documented in the corresponding log files and additional plots. Below are some selected configurations used for the presented results.

Additionally, aggregated and summarized results of the experiments are available in tabular form at the following link:

`https://docs.google.com/spreadsheets/d/1Uak1kR54UffO9Q2jjN1uvLxOFaKkG0_76U_Fm_ayMg8/edit?usp=sharing`

This table provides a compact overview of the key results and enables quick comparability of the different model configurations.

C

Confusion matrices

Since only one class is present during training, the model does not learn to generalize. As a result the entire column of any of the classes ”Normal”, ”DoS”, ”Probe”, ”R2L” or ”U2R” being filled with 1.

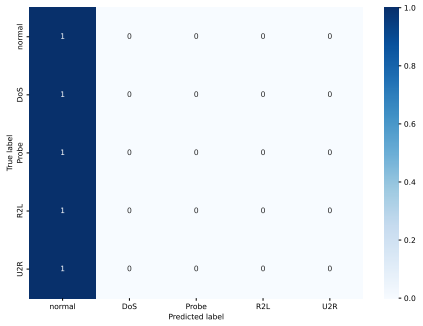

Figure C.1: Confusion matrix for training on a single class »Normal«.

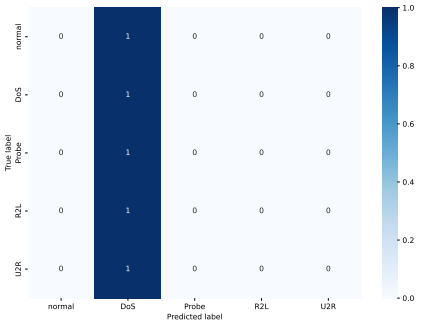

Figure C.2: Confusion matrix for training on a single class »DoS«.

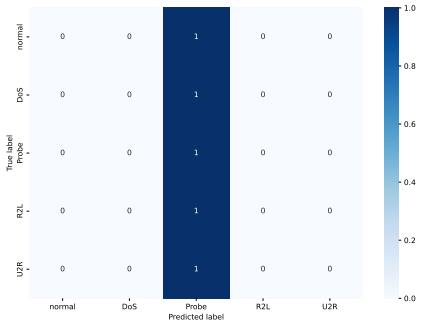

Figure C.3: Confusion matrix for training on a single class »Probe«.

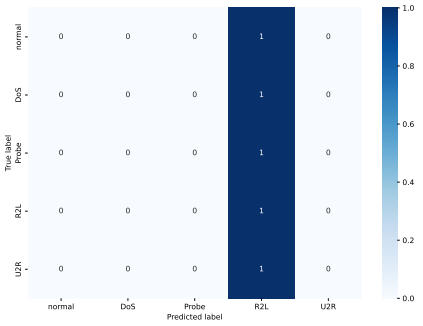

Figure C.4: Confusion matrix for training on a single class »R2L«.

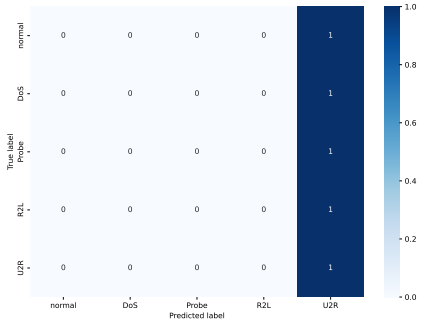

Figure C.5: Confusion matrix for training on a single class »U2R«.

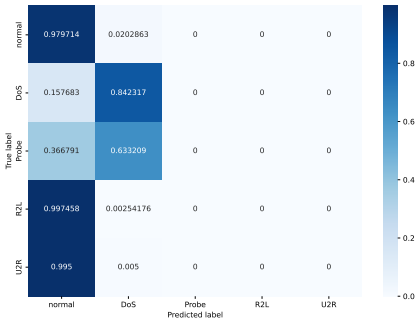

Figure C.6: Confusion matrix for training a model on ”Normal” and ”DoS”.

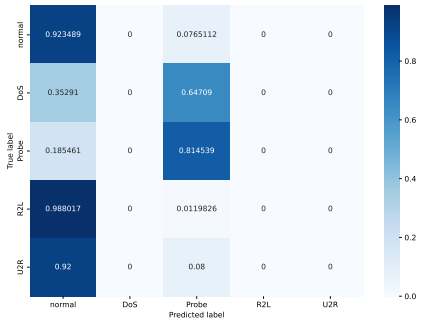

Figure C.7: Confusion matrix for training a model on ”Normal” and ”Probe”.

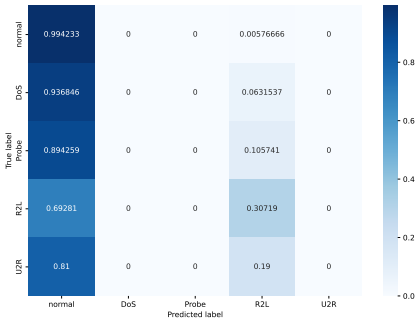

Figure C.8: Confusion matrix for training a model on ”Normal” and ”R2L”.

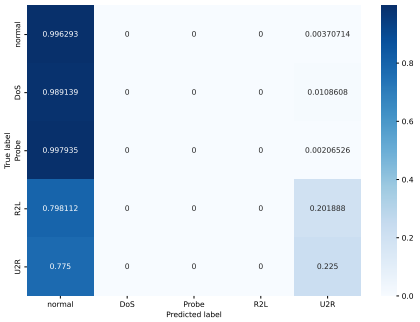

Figure C.9: Confusion matrix for training a model on ”Normal” and ”U2R”.

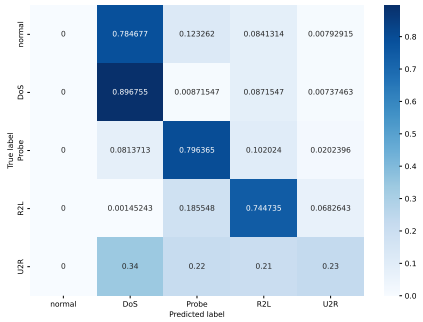

Figure C.10: Confusion matrix for training a model only on Attack types 1.

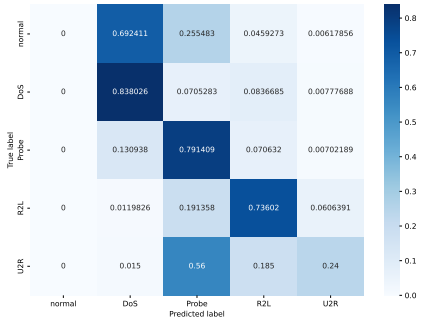

Figure C.11: Confusion matrix for training a model only on Attack types 2.

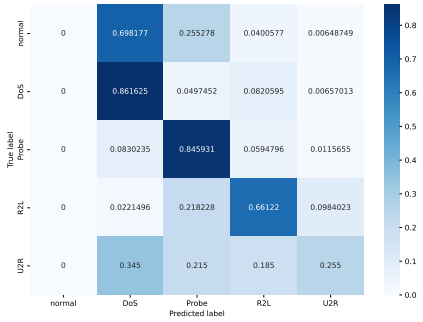

Figure C.12: Confusion matrix for training a model only on Attack types 3.

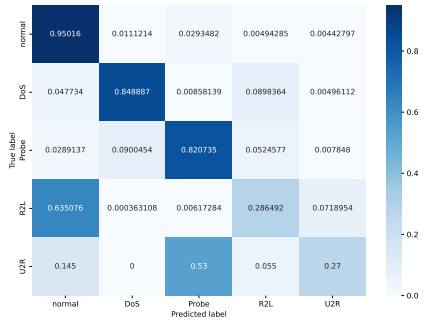

Figure C.13: Confusion matrix: Test results of the *All* model, which was trained on all classes.

C.1 Multiple Attackers

Confusion Matrices for training against Multiple Attackers

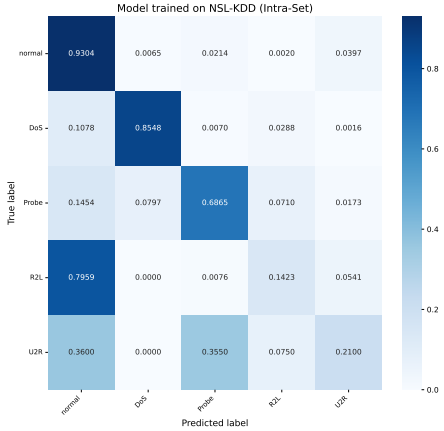

Figure C.14: Confusion matrix: Test results of the model which was trained against multiple attackers.

Distribution of Attacks in Training for Multiple Attackers

C.2 Distribution of Attacks in Training for Different Models

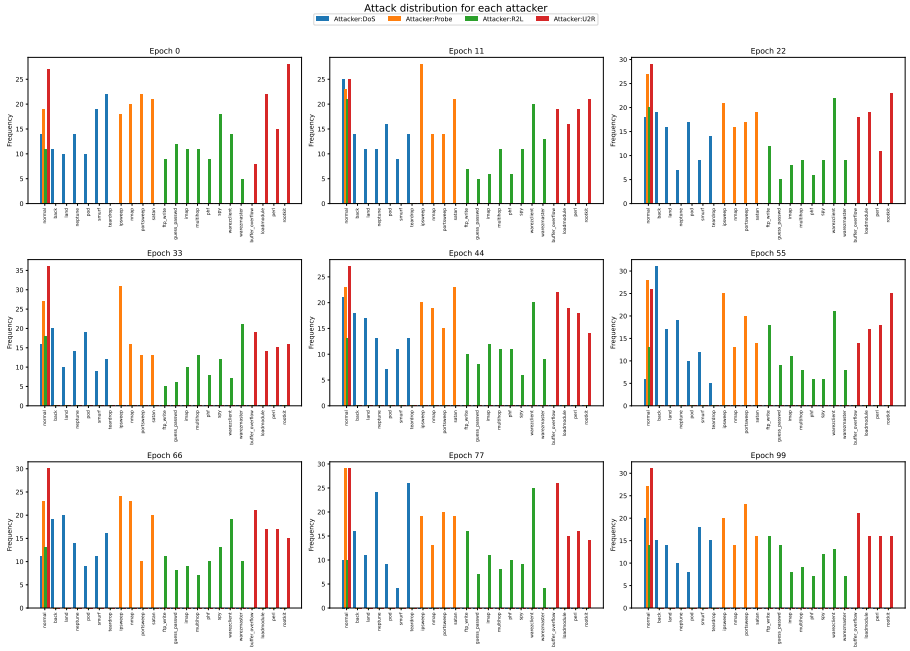

Figure C.15: Overview of the distribution of attacks presented to the multi-agent setup during training. The distribution is shown for each attacker agent, which is specialized in one of the four attack classes: DoS, Probe, R2L, and U2R. The histogram shows the number of samples selected by each attacker agent during training.

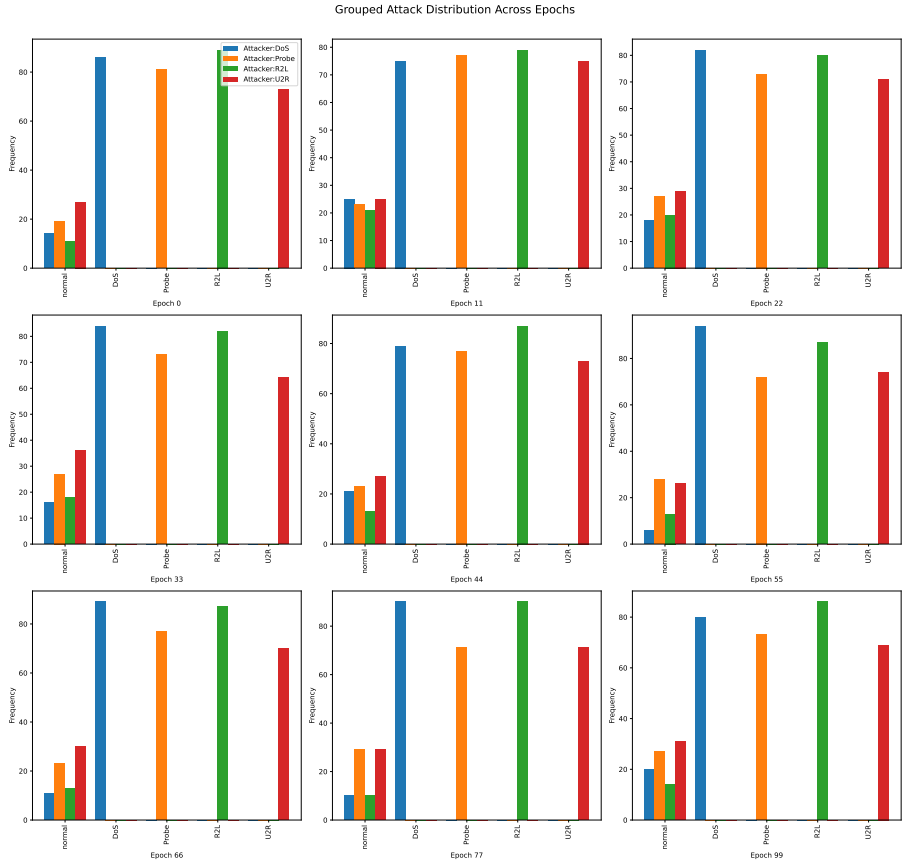

Figure C.16: Overview of the distribution of attack types presented to the multi-agent setup during training. The distribution is shown for each attacker agent, which is specialized in one of the four attack classes: DoS, Probe, R2L, and U2R. The barplots show the number of samples selected by each attacker agent during training.

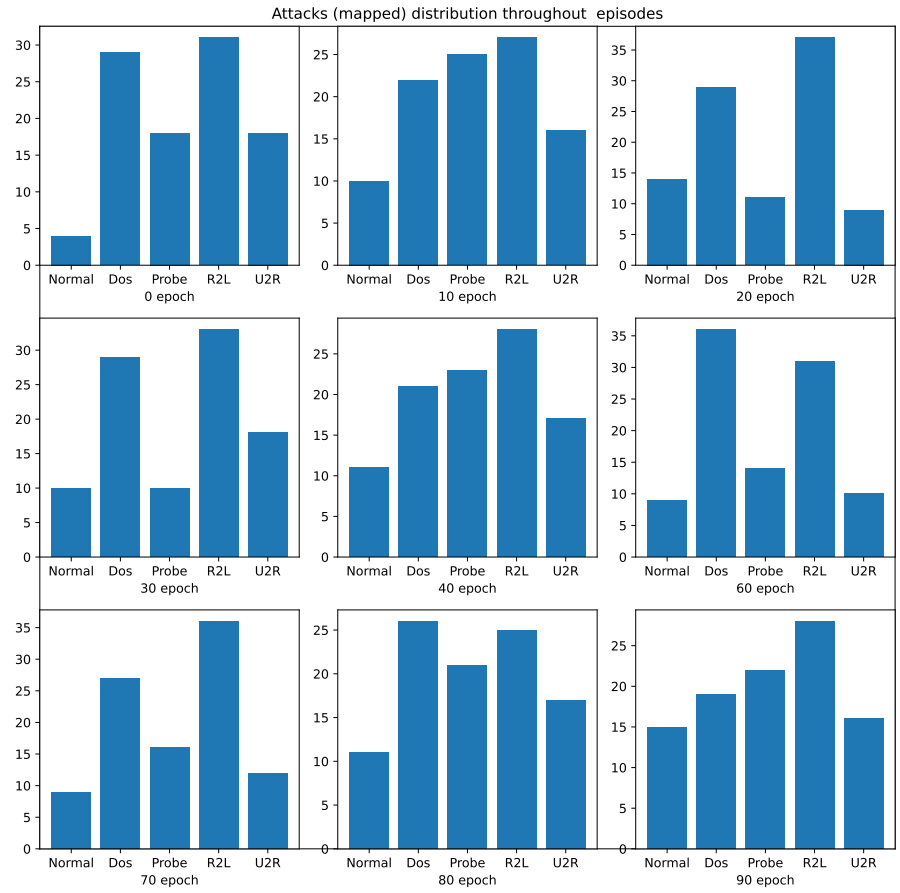

Figure C.17: Distribution of attacks during training a model with **4 hidden layers** for the attacker agent.

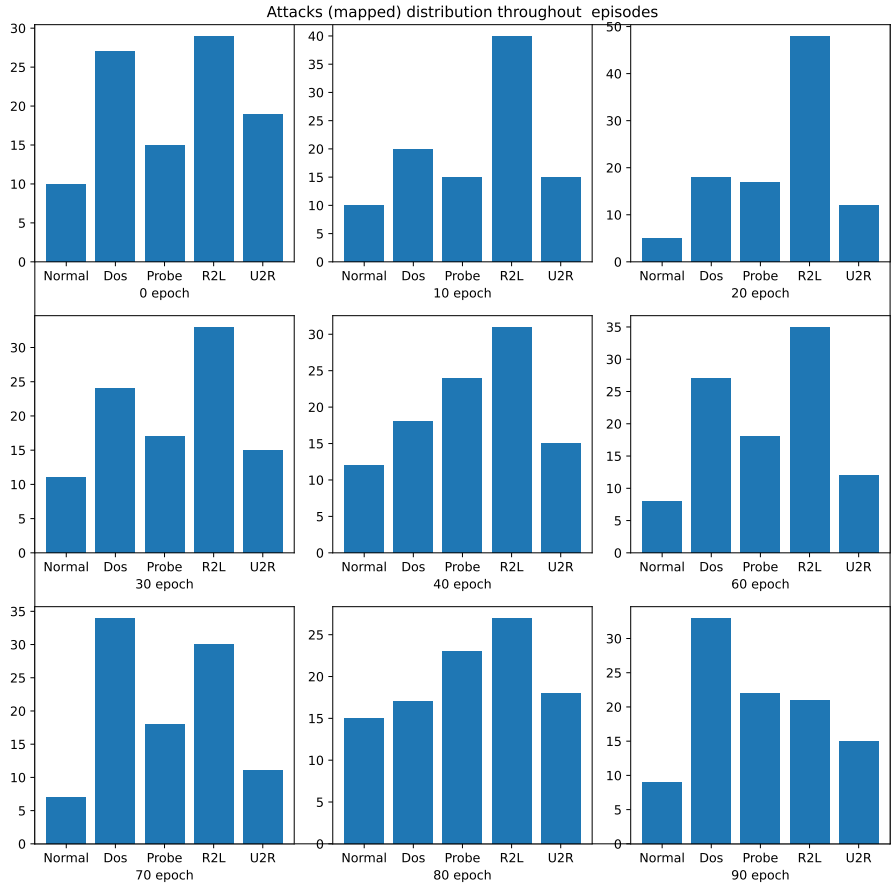

Figure C.18: Distribution of attacks during training a model with **8 hidden layers** for the attacker agent.

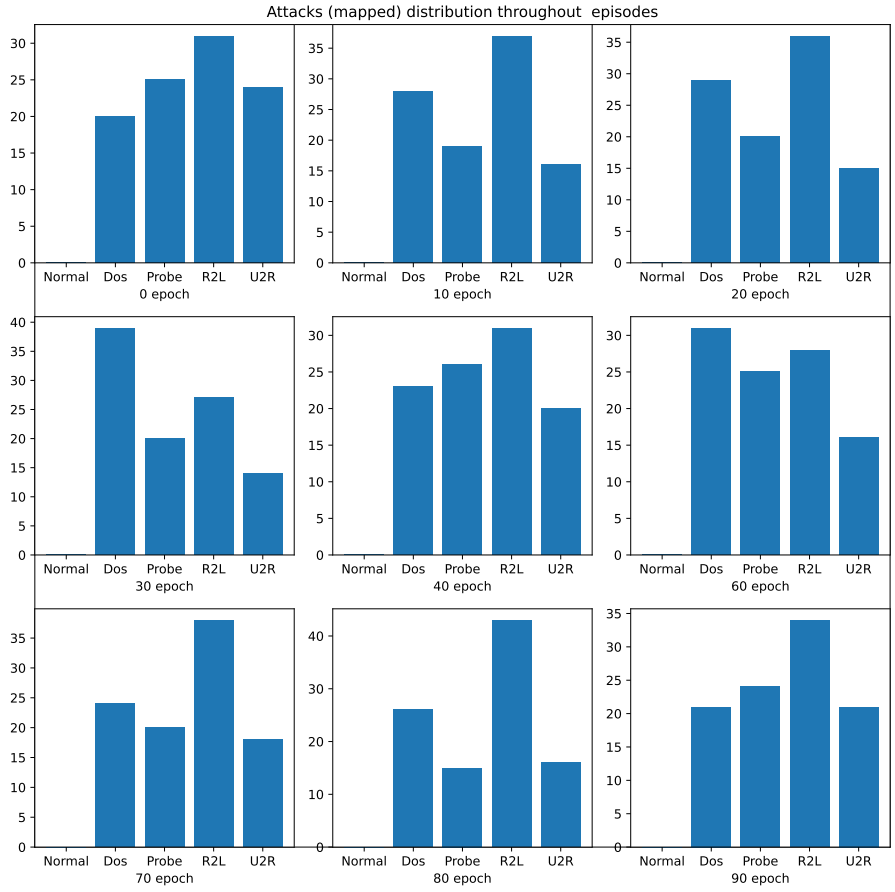

Figure C.19: Distribution of attacks during training a model with only attack types.

D

CIC-IDS Datasets

D.1 Attack Types and Distribution

In the following table the distribution of the different attack types in the CIC-IDS datasets is shown.

Table D.1: Attack classes and their number of occurrences in CIC-IDS-2017 and CSE-CIC-IDS-2018.

Attack Class	**Details**	**CIC-IDS-2017**	**CSE-CIC-IDS-2018**
		Count (Pct %)	Count (Pct %)
Benign	–	2,271,320 (80.32)	7,364,941 (84.59)
(D)DoS	Hulk	230,124 (13.43)	145,199 (11.17)
	DDOS	128,025	775,955
	GoldenEye	10,293	41,406
	DoS Slowloris	5,796	9,908
	Slowhttptest	5,499	43
Port Scan	–	158,804 (5.62)	–
Brute Force	FTP-Patator	7,935 (0.49)	53 (1.08)
	SSH-Patator	5,897	94,041
Web Attack	Brute Force	1,507 (0.08)	555 (0.01)
	XSS	652	227
	SQL Injection	21	79
Botnet	–	1,956 (0.07)	144,535 (1.66)
Infiltration	–	36 (<0.01)	129,786 (1.49)
Heartbleed	–	11 (<0.01)	–

Feature Details

The following table provides an overview of the features used in the CIC-IDS datasets after preprocessing. A more detailed description of the features can be found in the

'Towards Model Generalization for Intrusion Detection:Unsupervised Machine Learning Techniques' [21] or in the respective dataset documentation [15].

Table D.2: Overview of selected features after preprocessing.

#	Feature Name	Description
1	Packet Length Std	Standard deviation of packet length
2	Avg Packet Size	Average size of packets
3	Bwd Packet Length Max	Maximum length of backward packets
4	Packet Length Mean	Mean of all packet lengths
5	RST Flag Count	Count of RST flags in the flow
6	Fwd Header Length	Total length of headers in forward direction
7	Avg Bwd Segment Size	Average segment size in backward direction
8	Bwd Header Length	Total backward header length
9	Fwd Packet Length Max	Max forward packet length
10	Fwd Packet Length Mean	Mean forward packet length
11	Bwd Packets/s	Backward packets per second
12	Fwd Packets Length Total	Sum of all forward packet lengths
13	Total Backward Packets	Number of backward packets
14	Flow IAT Min	Minimum inter-arrival time in the flow
15	Packet Length Min	Minimum observed packet length
16	Packet Length Max	Maximum observed packet length
17	Fwd IAT Mean	Mean inter-arrival time in forward direction
18	URG Flag Count	Count of URG flags
19	Bwd Packet Length Std	Std of backward packet lengths
20	Total Fwd Packets	Total number of forward packets
21	Flow Bytes/s	Bytes transferred per second
22	Flow Duration	Total flow duration
23	Flow IAT Max	Maximum inter-arrival time in the flow
24	Bwd IAT Std	Std of backward IATs
25	Fwd Seg Size Min	Minimum segment size in forward direction
26	Packet Length Variance	Variance of all packet lengths
27	Bwd Packet Length Min	Minimum length of backward packets
28	Bwd IAT Min	Minimum backward inter-arrival time
29	Active Max	Maximum time the flow was active
30	Fwd Packets/s	Forward packets per second
31	Subflow Fwd Bytes	Bytes in forward subflow
32	Active Min	Minimum active time
33	Fwd IAT Min	Minimum forward inter-arrival time
34	Init Bwd Win Bytes	Initial backward window size
35	Idle Max	Maximum idle time
36	SYN Flag Count	Number of SYN flags in the flow

Continued on next page

Table D.2 – continued from previous page

#	Feature Name	Description
37	Label	Ground truth class label
38	Idle Mean	Mean idle time
39	Fwd Packet Length Min	Minimum length of forward packets
40	Bwd IAT Max	Maximum backward inter-arrival time
41	Bwd IAT Mean	Mean backward inter-arrival time
42	Fwd Packet Length Std	Std of forward packet lengths
43	Fwd IAT Total	Total forward inter-arrival time
44	Fwd IAT Std	Std of forward IATs
45	Bwd Packet Length Mean	Mean of backward packet lengths
46	Bwd IAT Total	Total backward inter-arrival time
47	FIN Flag Count	Number of FIN flags in the flow
48	Subflow Fwd Packets	Number of forward subflow packets
49	Subflow Bwd Bytes	Bytes in backward subflow
50	ECE Flag Count	Count of ECE flags
51	Avg Fwd Segment Size	Average forward segment size
52	Down/Up Ratio	Ratio of downlink to uplink traffic
53	Active Mean	Mean time the flow was active
54	Bwd Packets Length Total	Total length of backward packets
55	Flow IAT Std	Std of flow IATs
56	ACK Flag Count	Count of ACK flags in the flow
57	Subflow Bwd Packets	Number of packets in backward subflow
58	Init Fwd Win Bytes	Initial forward window size
59	Active Std	Std of active times
60	Protocol	Protocol used (e.g., TCP, UDP)
61	Flow Packets/s	Packets per second in the flow
62	Fwd PSH Flags	PSH flags in forward direction
63	PSH Flag Count	Total count of PSH flags
64	Idle Min	Minimum idle time
65	Flow IAT Mean	Mean inter-arrival time in the flow
66	Fwd IAT Max	Maximum forward inter-arrival time
67	Idle Std	Std of idle times
68	Fwd Act Data Packets	Number of active forward data packets

E

Boruta Feature Selection

This appendix presents the results of the Boruta feature selection procedure applied to the NSL-KDD dataset. The Boruta algorithm is a robust wrapper method that identifies all features relevant to the prediction task by comparing their importance to that of randomized shadow features.

The visual results are presented in three separate plots:

- The first plot displays the **confirmed features**, i.e., those that were consistently identified as important.
- The second plot shows the **rejected features**, which were consistently deemed irrelevant.
- The third plot contains the **tentative features**, whose importance could not be determined conclusively and may require further analysis.

In addition, the following sections list the confirmed features for the two main feature sets used in the evaluation: the compact *Core Features* and the extended *General Features*. These feature sets served as the basis for the experiments described in Chapter 6.

E.1 Confirmed Features from Boruta Analysis

Core Features (24 features)

- duration, src_bytes, dst_bytes, hot, logged_in, is_guest_login
- count, srv_count, same_srv_rate, diff_srv_rate, srv_diff_host_rate
- dst_host_count, dst_host_srv_count, dst_host_same_srv_rate
- dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate
- tcp, udp, ftp_data, http, other, telnet, SF

General Features (42 features)

- duration, src_bytes, dst_bytes, wrong_fragment, hot, num_failed_logins, logged_in, is_guest_login
- count, srv_count, serror_rate, rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate

- dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate
- dst_host_same_src_port_rate, dst_host_srv_diff_host_rate
- icmp, tcp, udp, domain_u, eco_i, ecr_i, finger, ftp, ftp_data
- http, other, private, smtp, telnet, urp_i, REJ, RSTO, RSTOS0, RSTR, SF, SH

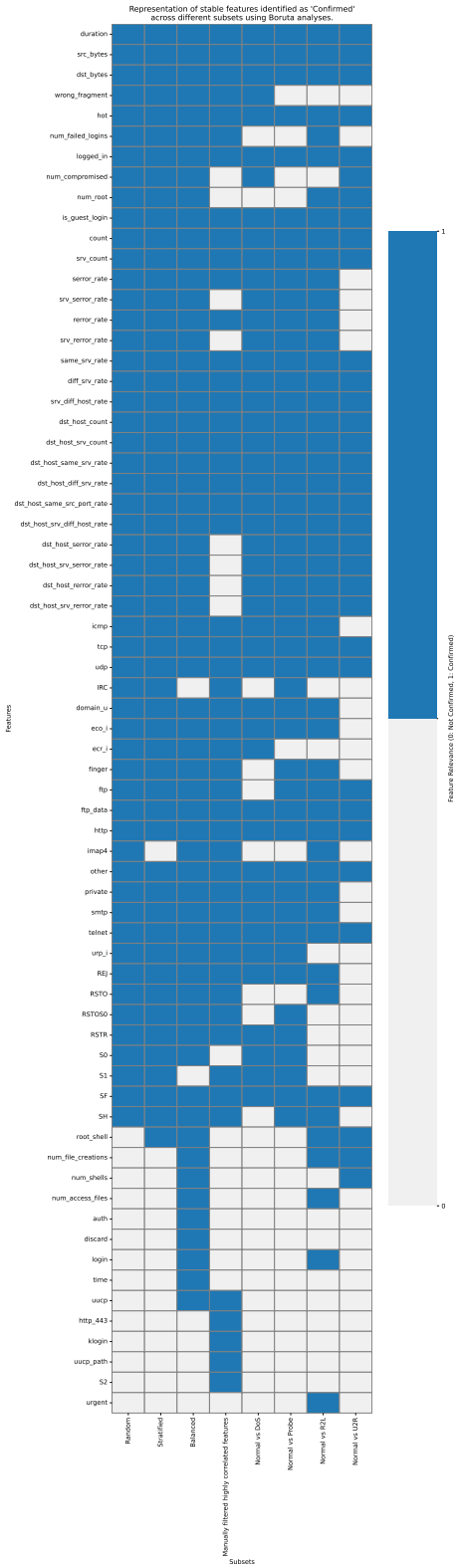

Figure E.1: Boruta feature selection results for **confirmed** features.

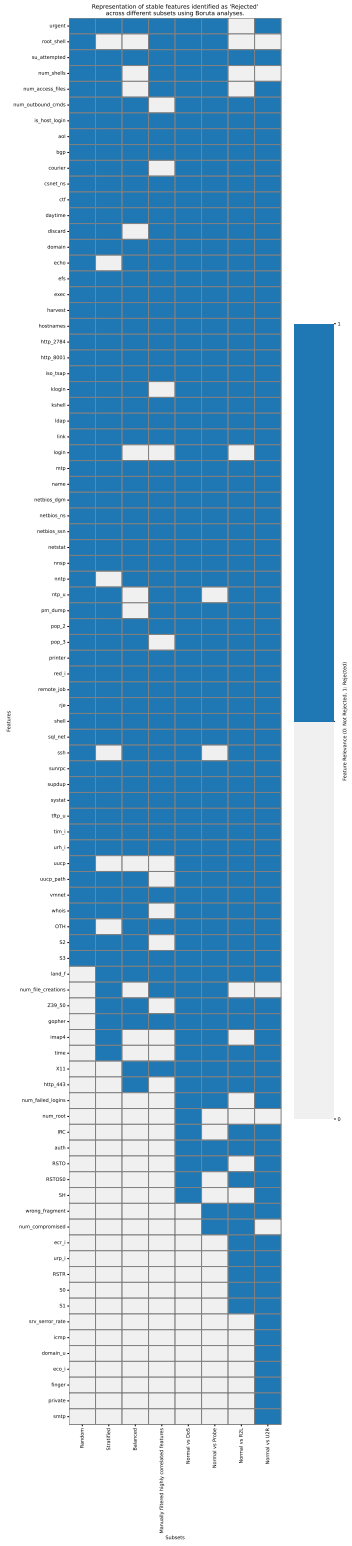

Figure E.2: Boruta feature selection results for **rejected** features.

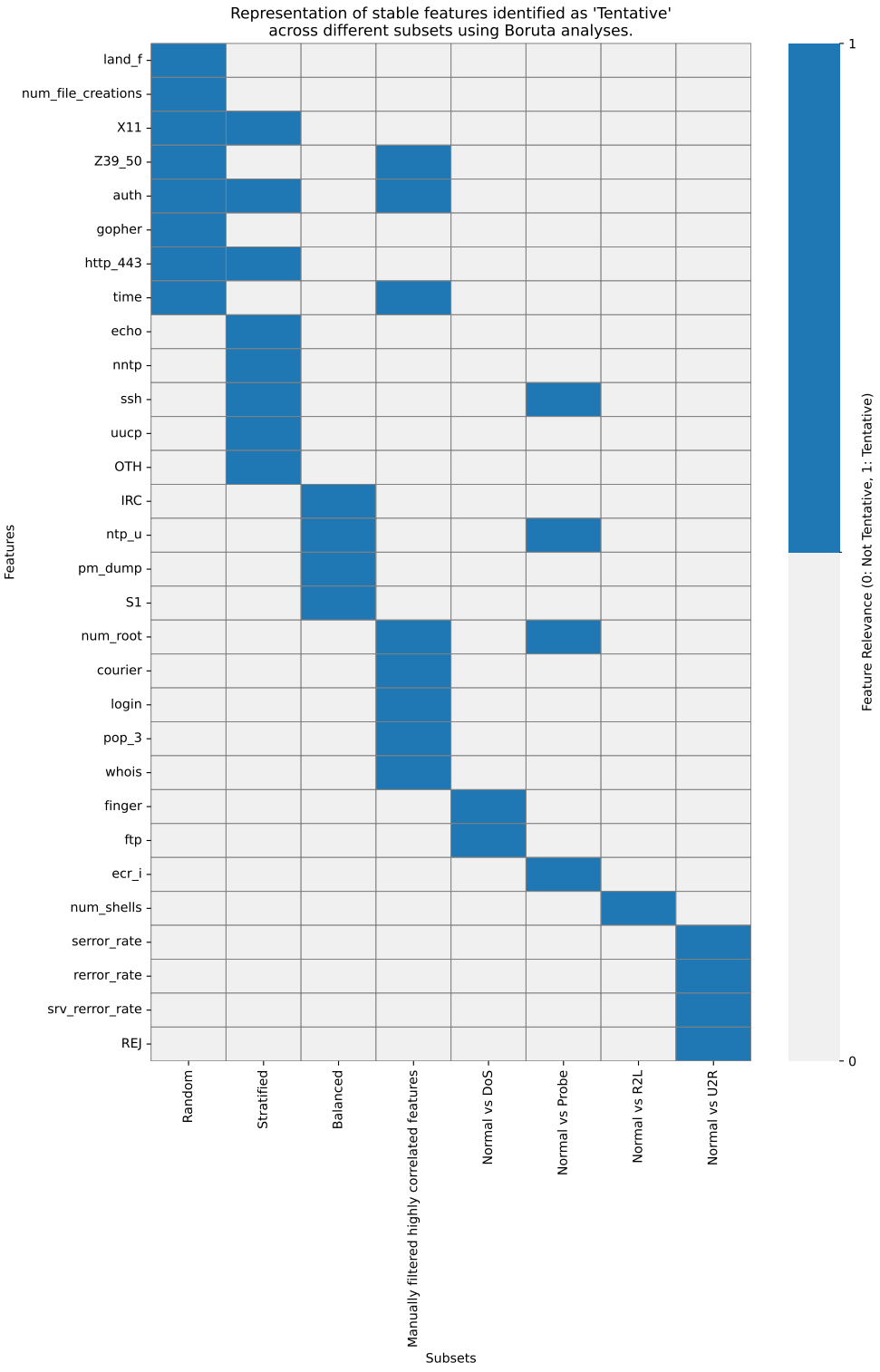

Figure E.3: Boruta feature selection results for **tentative** features.