People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
YAHIA FARES UNIVERSITY OF MEDEA

Faculty of Sciences

Department of Mathematics and Computer Science

Final Master Project

Branch: Computer science

Specialty: Systems Engineering and Web Technologies

Deep Q-Learning based approach for DoS attacks detection

Prepared and presented by:
Haitham Attab
Amine Takdenti

Supervisor:
Prof. Brahim Sahmadi

To my family, for their endless love and support.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Brahim Sahmadi, for their invaluable guidance and unwavering support throughout this research.

I also wish to thank [mention other people, e.g., colleagues, friends, family] for their encouragement and insightful discussions.

This work was made possible by [mention any funding or support].

Abstract

With the growing frequency and sophistication of Denial-of-Service (DoS) attacks, robust and adaptive detection mechanisms have become critical to maintaining the availability and reliability of computer networks. These attacks can severely disrupt essential services and lead to substantial financial losses, highlighting the need for intelligent and resilient defense strategies.

This thesis introduces a novel intrusion detection system based on Reinforcement Learning (RL) to detect and mitigate DoS attacks. By framing network security as a sequential decision-making problem, the proposed approach enables an RL agent to learn optimal detection policies through continuous interaction with a simulated network environment. The agent incrementally refines its ability to differentiate between benign and malicious traffic by leveraging a reward-based feedback mechanism that reinforces accurate classifications and penalizes misclassifications.

Key traffic features are systematically extracted and provided as state inputs to the RL agent, facilitating effective and dynamic behavioral analysis. This adaptive learning paradigm enhances the system's capability to recognize evolving attack patterns, ultimately contributing to a more proactive and resilient network defense against DoS threats.

ملخص

ضرورية والمتكيفة المتقدمة الكشف آليات أصبحت ،(DoS) الخدمة حجب بهجمات وتعقيد وتيرة تزايد مع
مالية خسائر في وتتسبب الحيوية الخدمات تعطل أن الهجمات لهذه يمكن وموثوقيتها. الشبكات توافر لضمان
ومرنة. ذكية دفاع استراتيجيات إلى الحاجة يُبرز مما كبيرة،
لاكتشاف (Reinforcement Learning) المعزز التعلم على يعتمد جديد تسلل كشف نظام المذكرة هذه تقدم
العميل يتمكن متسلسلة، قرارات اتخاذ كمشكلة الشبكات أمن تصوير خلال من لها. والتصدي DoS هجمات
محاكاة. شبكة بيئة مع المستمر التفاعل خلال من المثلى الكشف سياسات تعلم من المعزز التعلم في (Agent)
على قائم راجعة تغذية نظام باستخدام والخبيثة السليمة المرور حركة بين التمييز على تدريجياً قدرته بتحسين ويقوم
الخاطئة. التصنيفات ويعاقب الصحيحة التصنيفات يعزز المكافآت،
تحليل من يمكنه مما للعميل، حالة كمدخلات وتقديمها منهجي بشكل الأساسية المرور حركة ميزات استخراج يتم
أنماط على التعرف على النظام قدرة تعزيز في التكيفية التعلمية المنهجية هذه تسهم وديناميكية. بفعالية السلوك
DoS. تهديدات ضد واستباقية مرونة أكثر شبكي دفاع إلى النهاية في يؤدي مما المتغيرة، الهجمات

Résumé

Face à la fréquence et à la complexité croissantes des attaques par déni de service (DoS), la mise en place de mécanismes de détection robustes et adaptatifs est devenue essentielle pour garantir la disponibilité et la fiabilité des réseaux informatiques. Ces attaques peuvent perturber gravement les services critiques et engendrer des pertes financières importantes, soulignant ainsi la nécessité de stratégies de défense intelligentes et résilientes.

Ce mémoire propose un système de détection d'intrusion innovant basé sur l'apprentissage par renforcement (Reinforcement Learning) pour détecter et atténuer les attaques DoS. En formulant la sécurité réseau comme un problème de prise de décision séquentielle, notre approche permet à un agent d'apprentissage de découvrir des politiques de détection optimales via une interaction continue avec un environnement réseau simulé. L'agent améliore progressivement sa capacité à distinguer le trafic bénin du trafic malveillant grâce à un système de rétroaction basé sur des récompenses, qui renforce les classifications correctes et pénalise les erreurs.

Les principales caractéristiques du trafic sont extraites de manière systématique et utilisées comme entrées d'état pour l'agent, facilitant une analyse comportementale efficace et dynamique. Ce paradigme d'apprentissage adaptatif améliore la capacité du système à reconnaître les schémas d'attaque en évolution, contribuant ainsi à une défense réseau plus proactive et résiliente contre les menaces DoS.

Contents

List of Figures

List of Tables

List of Acronyms

AI   Artificial Intelligence

CIA   Confidentiality, Integrity, and Availability

CIC   Canadian Institute for Cybersecurity

CNN Convolutional Neural Network

CPU Central Processing Unit

CSV Comma-Separated Values

DDoS Distributed Denial-of-Service

DNS Domain Name System

DoS   Denial-of-Service

DQN Deep Q-Network

FTP   File Transfer Protocol

HIDS Host-based Intrusion Detection System

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

ICMP Internet Control Message Protocol

IDS   Intrusion Detection System

IP   Internet Protocol

IoT   Internet of Things

ML   Machine Learning

MLP   Multi-Layer Perceptron

NIDS Network-based Intrusion Detection System

NTP   Network Time Protocol

PCAP Packet Capture

RL   Reinforcement Learning

SIEM Security Information and Event Management

SSDP Simple Service Discovery Protocol

SVM Support Vector Machine

TCP Transmission Control Protocol

UDP User Datagram Protocol

General Introduction

Intrusion Detection Systems and Network Attacks

In the evolving landscape of cybersecurity, networks are increasingly vulnerable to a wide range of malicious activities that threaten their integrity, confidentiality, and availability. Among the most disruptive are **Denial-of-Service (DoS)** and its more severe counterpart, **Distributed Denial-of-Service (DDoS)** attacks. These assaults aim to overwhelm servers or network infrastructure by flooding them with traffic or exploiting protocol weaknesses, effectively rendering services inaccessible to legitimate users.

To detect and respond to such threats, organizations rely heavily on **Intrusion Detection Systems (IDS)**. IDS are security tools designed to monitor, analyze, and report on malicious activity within network traffic or system behavior. Broadly, IDS can be categorized into:

- **Signature-based IDS:** These systems rely on predefined patterns of known attacks (signatures). They are fast and accurate for known threats but completely blind to new, unknown, or obfuscated attacks.

- **Anomaly-based IDS:** These systems attempt to detect abnormal behaviors by building a profile of "normal" activity and flagging deviations. While they can identify novel threats, they frequently generate false positives, especially in dynamic or complex environments.

- **Hybrid IDS:** These systems aim to combine the strengths of both signature-based and anomaly-based methods to enhance overall detection performance.

Despite their widespread use, traditional IDS face several limitations:

- **Lack of adaptability:** Most systems require manual updates or retraining to detect evolving threats.

- **Poor real-time responsiveness:** Many solutions operate with a delay, reacting only after an attack is underway.

- **Scalability issues:** High-volume traffic or large-scale deployments can overwhelm static or rule-based detection mechanisms.

As cyber threats continue to evolve in complexity and scale, the limitations of existing IDS approaches highlight the urgent need for more intelligent, autonomous, and adaptive detection methods.

Context and Motivation

In the contemporary digital era, the reliability and continuous availability of network services form the backbone of global commerce, communication, and critical infrastructure. A primary threat to this stability is the Denial-of-Service (DoS) attack, and its more potent variant, the Distributed Denial-of-Service (DDoS) attack. These malicious activities aim to render online services and network resources unavailable to legitimate users by overwhelming them with an insurmountable volume of traffic or by exploiting protocol vulnerabilities. As attackers devise increasingly sophisticated and large-scale attacks, the need for intelligent, adaptive, and robust defense mechanisms has become more critical than ever.

Traditional Intrusion Detection Systems (IDS) have served as the first line of defense. However, they often rely on static, signature-based methods that, while effective against known threats, are inherently incapable of identifying novel or zero-day attacks. Anomaly-based systems, which attempt to solve this by creating a baseline of "normal" behavior, frequently suffer from high false positive rates, leading to operational friction and alert fatigue. Furthermore, conventional machine learning classifiers, though an improvement, often operate as static models that require periodic retraining and struggle to adapt in real-time to the dynamic and adversarial nature of cybersecurity threats.

This clear gap in defensive capabilities motivates the exploration of more advanced paradigms. Reinforcement Learning (RL), a subfield of machine learning, offers a compelling alternative. Unlike supervised learning, RL focuses on training an autonomous agent to make optimal decisions through interaction with an environment, guided by a system of rewards and penalties. This approach is inherently adaptive, allowing the agent to learn and refine its strategies over time. The potential to create a self-improving, context-aware detection system that can respond dynamically to evolving threats provides the core motivation for this research.

Problem Statement

The central problem addressed by this thesis is the inadequacy of traditional and static machine learning-based Intrusion Detection Systems in providing adaptive, real-time defense against modern, sophisticated DDoS attacks. These existing systems exhibit several key limitations:

- **Lack of Adaptability:** Signature-based systems cannot detect new attack vectors, while static ML models fail to adapt to evolving attack patterns without manual retraining.

- **High False Positive Rates:** Anomaly detection systems often misclassify legitimate traffic spikes (flash crowds) as attacks, disrupting normal operations.

- **Reactive Nature:** Most systems are reactive, identifying attacks only after they have begun to impact the network, rather than learning to predict and preempt them.

This research seeks to answer the following question: **Can Reinforcement Learning, specifically a Deep Q-Network (DQN) approach, be effectively adapted to the supervised classification task of DDoS detection to create a more dynamic, accurate, and autonomous system?**

Objectives

To address the problem statement, this thesis sets out the following primary objectives:

1. To conduct a thorough review of Denial-of-Service attacks, traditional Intrusion Detection Systems, and the foundational principles of Reinforcement Learning to establish a strong theoretical basis.

2. To design and develop a novel detection framework that reframes the DDoS classification task as a sequential decision-making problem, solvable with a Deep Q-Network (DQN) agent.

3. To implement two distinct neural network architectures for the DQN's Q-function approximator—a Multi-Layer Perceptron (MLP) and a Convolutional Neural Network (CNN)—to explore different feature representation capabilities.

4. To train and rigorously evaluate the proposed models using a modern, comprehensive benchmark dataset (CIC-DDoS2019), measuring performance based on key metrics such as accuracy, precision, recall, and F1-score.

5. To compare the performance of the MLP-based and CNN-based DQN agents to determine the most effective architecture for this task.

Overall Methodology

The research will be conducted through a structured, multi-stage process:

1. **Data Preprocessing:** The CIC-DDoS2019 dataset will be acquired and subjected to a comprehensive preprocessing pipeline. This includes data cleaning, binary label transformation, feature selection using a Random Forest classifier to identify the most salient features, Min-Max normalization to scale the data, and random undersampling to create a balanced dataset for training.

2. **Model Design:** A custom Reinforcement Learning environment will be simulated from the static dataset. A Deep Q-Network (DQN) agent will be designed, defining the state space (from network features), action space (classify as 'Benign' or 'Attack'), and a reward function that provides feedback based on classification correctness.

3. **Architectural Implementation:** Two Q-network architectures will be implemented in Python using deep learning libraries. The first will be a standard Multi-Layer Perceptron. The second will be a Convolutional Neural Network, which requires reshaping the one-dimensional feature vector into a two-dimensional "feature-image" to exploit spatial correlations between features.

4. **Training and Evaluation:** Both models will be trained using a novel loop that blends supervised signals (the true labels) with RL principles (value estimation of future states). The trained models will be evaluated on a held-out test set to assess their detection performance and generalization capabilities.

Thesis Structure

This thesis is organized into six chapters, each building upon the last to present a complete account of the research:

- **Chapter 1: Intrusion Detection Systems (IDS)** lays the theoretical groundwork by discussing fundamental security concepts, detailing the various types of DoS/DDoS attacks, and explaining the architecture and limitations of traditional IDS.

- **Chapter 2: Reinforcement Learning for Cybersecurity** introduces the core concepts of Reinforcement Learning, with a specific focus on Deep Q-Learning. It explores how the RL paradigm can be modeled for security tasks, covering state, action, and reward design.

- **Chapter 3: Architectural Blueprint and Implementation** presents the technical core of this work. It details the data preprocessing pipeline and the architectural design of the two proposed DQN agents (MLP and CNN-based).

- **Chapter 4: Experiments and Results** will describe the experimental setup, present the performance results of the trained models, and provide a comparative analysis and discussion of the findings.

Chapter 1

Intrusion Detection System (IDS)

1.1 Security Fundamentals

Introduction

Security fundamentals refer to the essential principles, concepts, and practices that form the foundation of information security. These fundamentals encompass a wide range of technical and organizational measures aimed at protecting sensitive information and systems from unauthorized access, theft, damage, or other forms of compromise.

Key security fundamentals include Confidentiality, Integrity, Availability, Authentication, Authorization, Encryption, Risk Management, Incident Response, and Disaster Recovery. Together, these principles establish the basis of a comprehensive information security program, enabling organizations to effectively safeguard their critical information assets and maintain the trust of stakeholders.[1]

1.1.1 Core Security Concepts

Network security encompasses strategies and technologies to protect systems from cyber threats, particularly Denial of Service (DoS) attacks that aim to disrupt service availability. Several core principles serve as the foundation for secure systems.

CIA Triad

Confidentiality Confidentiality ensures that sensitive information is only accessible to authorized users. Techniques such as encryption, user authentication, and access control policies prevent unauthorized data access. While DoS attacks do not typically aim to breach confidentiality directly, successful exploitation may lead to indirect confidentiality violations if attackers cause service misconfigurations or force failovers to insecure states.

Integrity Integrity guarantees that data is accurate and unaltered. It is maintained through cryptographic hash functions, checksums, and digital signatures that validate whether data has been tampered with. During a DoS attack, integrity may be compromised by interrupting legitimate updates or corrupting processes due to system overload.

Availability Availability ensures that services and systems remain accessible to legitimate users at all times. This principle is the primary target of DoS attacks, which flood networks or services with excessive requests, causing slowdowns or complete denial of access. Maintaining availability involves redundancy, load balancing, and proactive mitigation strategies like rate limiting and firewalls.

Figure 1.1: Principles of information security

Extended Security Principles

Non-repudiation Non-repudiation guarantees that an entity cannot deny having performed a particular action, such as sending a message or initiating a transaction. This is enforced through digital signatures and secure logging mechanisms. In the context of DoS attacks, non-repudiation helps trace attack origins and supports legal accountability.

Authenticity Authenticity confirms that data, communications, or users are genuine and not forged. Authentication protocols, digital certificates, and cryptographic techniques are used to ensure that data comes from trusted sources. This is crucial for filtering legitimate traffic from spoofed attack traffic in DoS scenarios.

Accountability Accountability ensures that all actions within a system can be traced to responsible users or processes. It involves logging, auditing, and monitoring to track behavior. Accountability is vital for forensic analysis after a DoS attack and for strengthening defenses against future intrusions.

1.2 Denial-of-Service (DoS) Attacks

Introduction Denial-of-Service (DoS) attacks aim to make a system or network resource unavailable to its intended users by overwhelming it with excessive traffic or exploiting protocol-level vulnerabilities. These attacks can disrupt services, degrade performance, or completely shut down access. The most common types include:

1.2.1 Volumetric Attacks

Volumetric attacks aim to saturate a target's bandwidth by generating an overwhelming amount of traffic. This often involves amplification techniques or high-rate packet floods. Common examples include UDP floods and DNS amplification attacks, which leverage misconfigured servers to multiply traffic directed at the victim [3].

In a UDP flood, attackers send large numbers of spoofed UDP packets to random or specific ports, consuming bandwidth and processing power. DNS amplification uses small queries to open resolvers with the victim's IP address, causing them to return large responses to the victim.
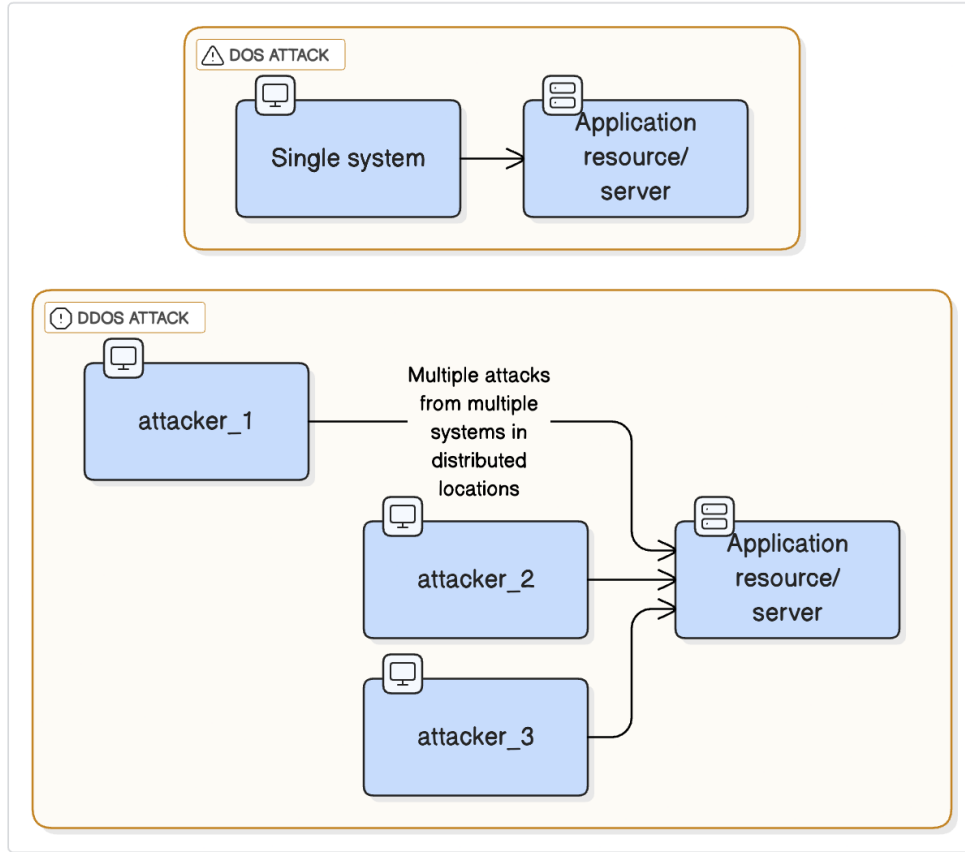
Figure 1.2: DoS and DDoS Attack Traffic Comparison

A notable example is the Mirai botnet, which infected hundreds of thousands of insecure IoT devices to coordinate massive traffic streams toward its targets [3]. The goal of volumetric attacks is to clog network links (Gbps or Tbps scale), preventing legitimate access.

1.2.2 Protocol Attacks

Protocol attacks exploit weaknesses in Layer 3/4 protocols (network or transport layer) to exhaust server or network resources. Unlike volumetric attacks, they do not require high bandwidth but instead consume stateful resources like connection tables or CPU cycles [3].

A classic example is the TCP SYN flood, where attackers send numerous SYN packets without completing the handshake, filling the server's connection queue [4]. ICMP floods and Ping of Death attacks exploit the Internet Control Message Protocol to crash or freeze systems by sending malformed or excessive traffic.

Other examples include fragmentation attacks (e.g., Teardrop), which send overlapping IP fragments to crash reassembly logic, and ACK/FIN floods that exhaust firewall state tables.

1.2.3 Application Layer Attacks

Application-layer (Layer 7) attacks generate traffic that appears legitimate at the application level but consumes excessive server resources such as CPU, memory, or threads [3]. These attacks are difficult to detect because they mimic real user behavior.

HTTP GET/POST floods can overwhelm web servers by triggering costly operations. A well-known example is Slowloris, which sends partial HTTP headers slowly to hold open many connections and exhaust the web server's pool [5]. Other examples include RUDY (R-U-Dead-Yet) attacks and HTTP floods targeting dynamic or database-backed content.

18

1.2.4 Distributed Denial-of-Service (DDoS)

DDoS attacks use multiple compromised machines (botnets) to launch coordinated attacks, making them harder to block and vastly more powerful than single-source DoS attacks [3].

Botnets like Mirai leverage IoT devices to simultaneously launch volumetric or protocol-based attacks. Reflective amplification (e.g., using open DNS/NTP servers) further increases traffic impact. Mitigation requires upstream filtering, rate-limiting, and often third-party scrubbing services.

1.2.5 Network Traffic Analysis

Mitigating DoS/DDoS attacks relies on monitoring and anomaly detection. This involves establishing a baseline of normal network behavior and identifying deviations in volume, protocol use, or source IPs [3, 6].

Anomaly-based intrusion detection systems (IDS) can flag unusual spikes, connection patterns, or TCP flag anomalies. Signature-based systems match known attack patterns. Flow analysis tools (e.g., NetFlow, sFlow) help detect irregular byte/packet rates or unusual source-destination pairs.

Advanced methods include machine learning to classify deviations. Once detected, defenses such as rate limiting, traffic shaping, or redirection to mitigation services are employed to maintain availability.

1.3 Intrusion Detection Systems (IDS)

An **Intrusion Detection System (IDS)** is a cybersecurity solution designed to monitor network or system activities for malicious actions or policy violations. Upon detecting such activities, the IDS typically alerts system administrators or integrates with centralized security tools like Security Information and Event Management (SIEM) systems to facilitate a coordinated response [7].

1.3.1 IDS Architecture

Monitored System Feedback Flow
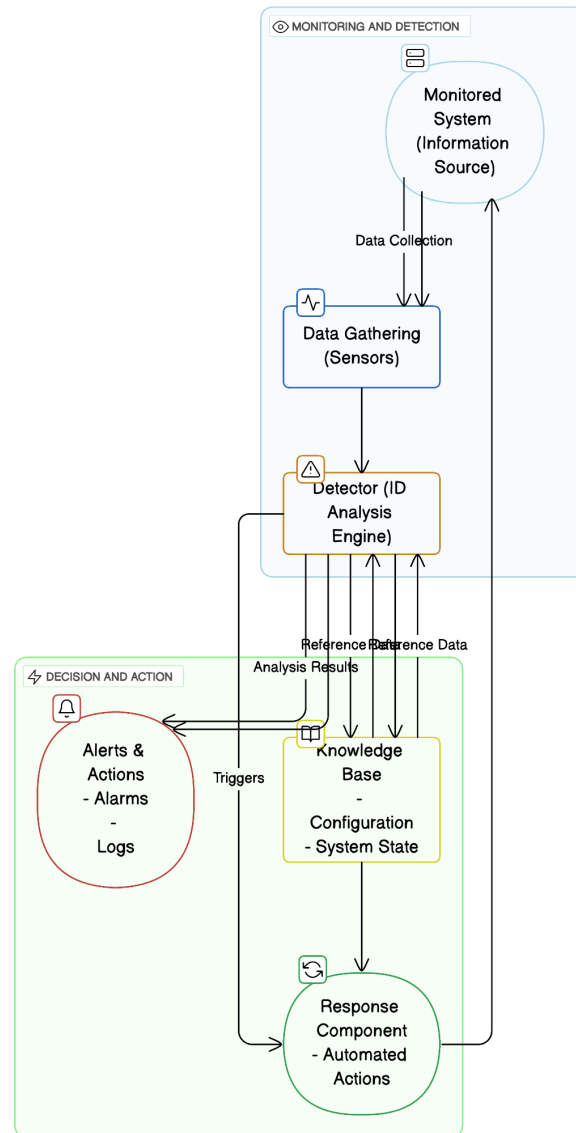
Figure 1.3: Network-based Intrusion Detection System (NIDS) Architecture

Modern IDS platforms implement modular architectures with several key components:

- **Sensors:** are the front line of an IDS, deployed at strategic network chokepoints—such as mirror (SPAN) ports, network taps, or virtual interfaces in cloud environments—to collect raw traffic data.

- **Analysis Engines:** form the IDS's "brain," taking the sensor-collected data and applying a mixture of rule-based and intelligent techniques to detect threats.

- **Knowledge Base:** underpins all detection logic by maintaining up-to-date signatures, behavioral profiles, and historical datasets.

- **Response Systems:** close the loop between detection and defense. Once the analysis engine assigns a confidence score to an event, the response component generates alerts.

For DOS attack detection, these components must operate with high efficiency and scale to process enormous traffic volumes during attack scenarios.

1.3.2   IDS Types and Functionality

Intrusion Detection Systems serve as the primary monitoring and detection layer for network security threats, including DOS attacks:

- **Network-based IDS (NIDS):** A NIDS is a monitoring solution placed at key junctions within a network to passively capture and inspect all passing traffic.
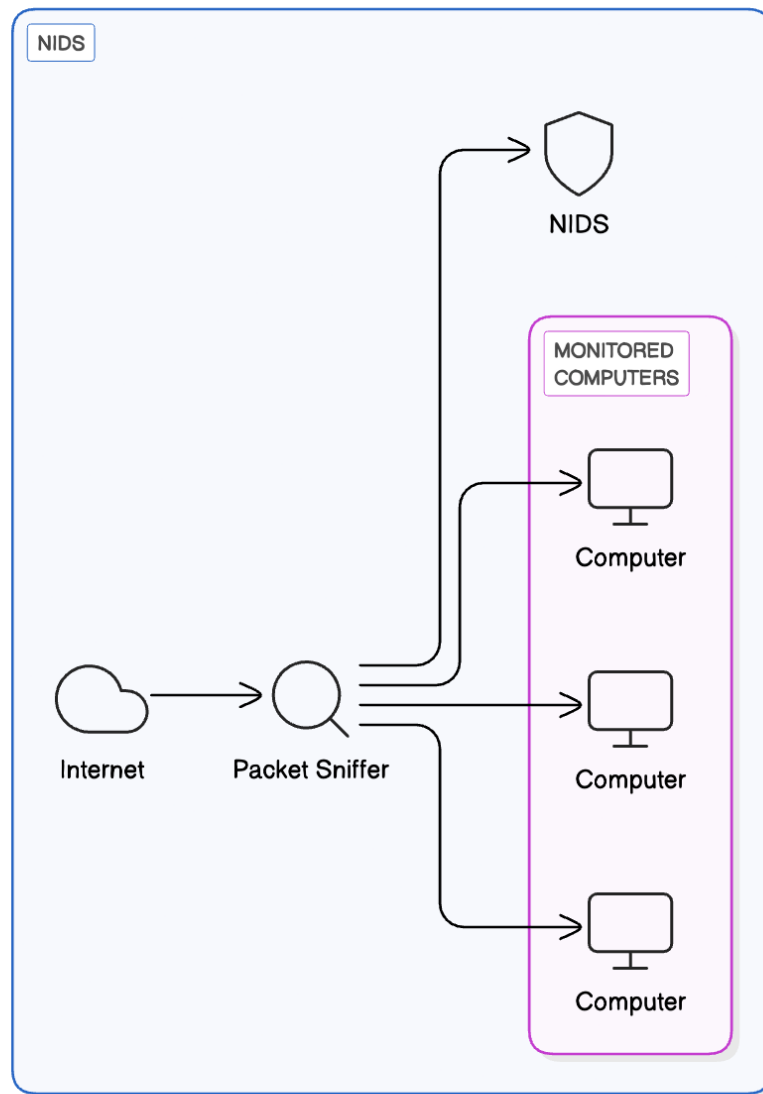
Figure 1.4: Network-based Intrusion Detection System (NIDS) Diagram

- **Host-based IDS (HIDS):** A HIDS operates at the level of individual endpoints or servers, offering deep visibility into system-specific activities.
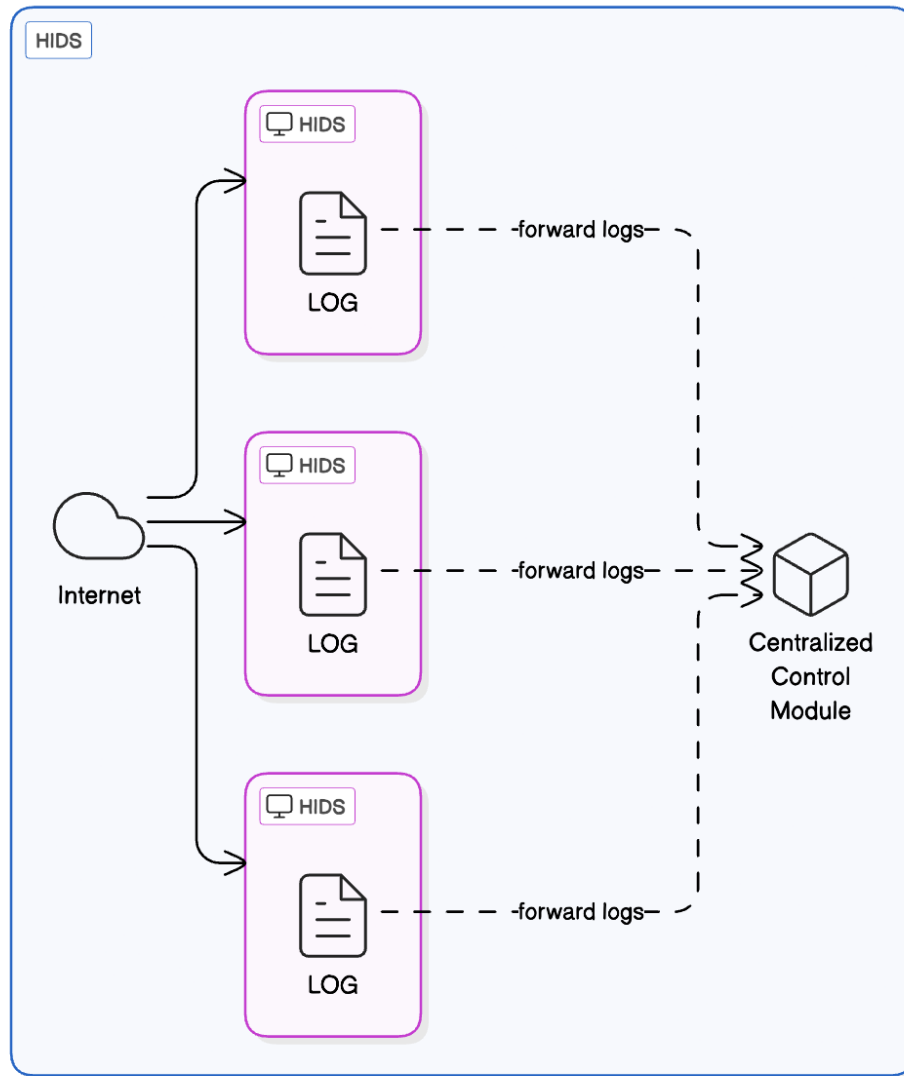
Figure 1.5: Host-based Intrusion Detection System (HIDS) Diagram

- **Detection Methodologies:**

  - **Signature-based:** Matches observed traffic against known attack patterns (signatures). Highly effective for known attacks but ineffective against zero-day attacks.
  - **Anomaly-based:** Establishes baselines of normal behavior and flags significant deviations. Can detect unknown attacks but may generate more false positives.
  - **Hybrid Systems:** Combine both methodologies to leverage their respective strengths and reduce false positives.

Limitations of Traditional IDS for DOS attack detection include difficulty processing high-volume traffic, distinguishing flash crowds from attacks, limited adaptation, high false-positive rates, and high resource consumption. These limitations necessitate AI-driven approaches like reinforcement learning.

1.4 Classification in Security Contexts

1.4.1 Classification Paradigms

Classification forms the foundation of automated threat detection: **Binary Classification:** Distinguishes between normal and attack traffic. **Multi-class Classification:** Distinguishes between multiple attack types. **Hierarchical Classification:** Structures threats in a tree-like format. **Multi-label Classification:** Assigns multiple categories to a single traffic flow.

1.5 Conclusion

This chapter provided a comprehensive overview of Intrusion Detection Systems (IDS) and their role in defending against Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) attacks. It began by establishing core security principles—confidentiality, integrity, and availability—and extended to non-repudiation, authenticity, and accountability, which form the foundational requirements for secure systems.

We then examined the nature of DoS attacks in detail, highlighting their categorization into volumetric, protocol, and application-layer threats, along with their increasingly distributed and complex manifestations. Through traffic analysis and classification paradigms, the chapter demonstrated how accurate identification of attack patterns is essential for timely defense.

Finally, we explored the architecture, types, and operational methodologies of IDS—including signature-based, anomaly-based, and hybrid systems—alongside their limitations in handling dynamic, large-scale attack scenarios.

Given the static and reactive nature of traditional IDS, there is a growing need for intelligent, adaptive systems capable of learning from evolving threats. This chapter sets the theoretical groundwork for such systems and motivates the transition toward machine learning and, more specifically, reinforcement learning techniques. The following chapters will build on this foundation by introducing Deep Q-Learning as a novel approach for dynamic and autonomous intrusion detection.

Chapter 2

Deep Q-Learning for Intrusion Detection

2.1 Introduction

This chapter establishes the theoretical foundation for the novel intrusion detection methodology proposed in this thesis. The objective is to move beyond traditional, static detection systems by leveraging the adaptive and autonomous capabilities of modern artificial intelligence. We begin by introducing the key concepts of Deep Learning (DL), which provides the powerful function approximation tools needed to handle complex, high-dimensional data.

Subsequently, we delve into the paradigm of Reinforcement Learning (RL), a field of machine learning focused on goal-oriented learning through interaction. We will outline the fundamental components of an RL system—the agent, environment, states, actions, and rewards—that form the basis of sequential decision-making.

The core of this chapter brings these two fields together in the form of Deep Q-Learning (DQN). We will explain how DQN combines the perceptive power of deep neural networks with the decision-making framework of Q-learning to solve complex problems with large state spaces. Finally, we discuss the application of these advanced techniques to the domain of Intrusion Detection Systems (IDS), setting the stage for the specific architectural designs and experiments detailed in the subsequent chapters.

2.2 Deep Learning: Key Concepts

Deep Learning (DL) is a subfield of machine learning inspired by the structure and function of the human brain, utilizing artificial neural networks with multiple layers—hence the term "deep." Unlike traditional machine learning models that often rely on manually engineered features, deep learning models are designed to learn hierarchical representations of data automatically. By processing data through successive layers, they can identify simple patterns at the initial layers and compose them into more complex and abstract patterns at deeper layers. This makes them exceptionally powerful for finding intricate structures in large, high-dimensional datasets, such as those found in network traffic analysis.
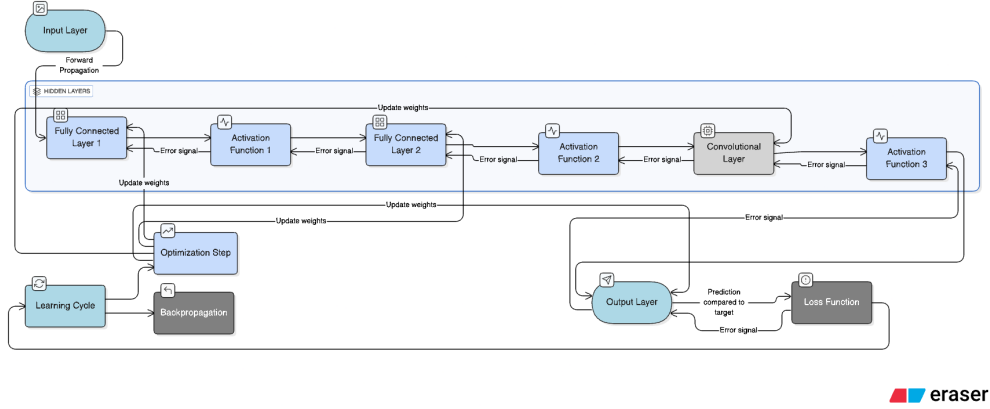
Figure 2.1: Fundamental structure of a deep neural network, showing the flow from input data through hidden layers to an output prediction.

A typical neural network, as illustrated in Figure 2.1, consists of an input layer, one or more hidden layers, and an output layer. Each layer contains interconnected nodes known as **neurons**. A neuron computes a weighted sum of its inputs, adds a bias term, and then passes this result through a non-linear **activation function**. The purpose of the activation function (e.g., ReLU, Sigmoid, Tanh) is crucial: it introduces non-linearity into the model, enabling the network to learn complex relationships that a purely linear model could not capture. The connections between neurons have associated **weights** and **biases**, which are the learnable parameters that the network adjusts during training to minimize its prediction error.

The process of training a deep neural network is an iterative optimization cycle aimed at finding the optimal set of weights and biases. This cycle generally involves the following steps:

1. **Forward Propagation:** Input data (e.g., a feature vector from a network flow) is fed into the network's input layer. The data then propagates forward through the hidden layers. At each layer, every neuron performs its computation, and its output becomes the input for the next layer. This process continues until a final prediction is generated at the output layer. This step is essentially the network making its "best guess" based on its current parameters.

2. **Loss Function:** The network's prediction is compared to the true value (the ground-truth label) using a **loss function**. The loss function quantifies the discrepancy or error between the prediction and the actual target. For classification tasks like ours, the **Cross-Entropy loss** is commonly used. A high loss value indicates a poor prediction, while a low loss value signifies a good one. The goal of training is to minimize this loss.

3. **Backpropagation:** This is the core mechanism for learning. Based on the calculated loss, the error is propagated backward through the network, from the output layer to the input layer. Using the chain rule from calculus, backpropagation efficiently calculates the gradient of the loss function with respect to each weight and bias in the network. This gradient essentially represents the contribution of each parameter to the total error—it tells the network "in which direction" and "by how much" each weight should be adjusted to reduce the error.

4. **Optimization:** An **optimization algorithm**, such as Adam or Stochastic Gradient Descent (SGD), uses the gradients computed during backpropagation to update the network's weights and biases. It adjusts the parameters in the opposite direction of their gradient, effectively taking a small step towards a lower loss. The size of this step is controlled by a hyperparameter called the **learning rate**.

This cycle of forward propagation, loss calculation, backpropagation, and optimization is repeated for many iterations (or "epochs") over the training dataset. With each iteration, the network's parameters are fine-tuned, and it gradually becomes better at making accurate predictions, converging towards a state of minimal error.

In this thesis, we utilize two primary deep learning architectures to serve as the Q-network for our agent:

Multi-Layer Perceptron (MLP)   The MLP is the quintessential deep learning model. It consists of one or more fully connected hidden layers, where every neuron in a given layer is connected to every neuron in the subsequent layer. It is a universal function approximator, well-suited for processing structured, tabular data where the input is a flat vector of features and there is no inherent spatial or temporal relationship between them.

Convolutional Neural Network (CNN)   Originally designed for image processing, CNNs excel at detecting spatial hierarchies and patterns in grid-like data. They use specialized layers with filters (or kernels) that slide across the input, detecting local patterns such as edges or textures. By stacking these layers, a CNN can learn to recognize increasingly complex features. As we will detail later, we adapt this powerful architecture to our tabular network data by reshaping the 1D feature vector into a 2D "feature-image," allowing the CNN to discover localized correlations between related network features.

2.3   Reinforcement Learning

Reinforcement Learning (RL) is a paradigm of machine learning where an autonomous **agent** learns to make optimal decisions by interacting with an **environment**. Unlike supervised learning, the agent is not given explicit correct answers. Instead, it receives feedback in the form of numerical **rewards** or penalties for its actions, and its goal is to learn a strategy, or **policy**, that maximizes its cumulative reward over time.

The core components of the RL framework are (see Figure 2.2):

- **Agent:** The learner and decision-maker. In our context, this is the intrusion detection model.

- **Environment:** The external world with which the agent interacts. For our task, this is the network traffic environment.

- **State (s_t):** A snapshot of the environment at a specific time step t. This could be a vector of features describing a network flow.

- **Action (a_t):** A choice made by the agent in a given state. For example, classifying the traffic as 'Benign' or 'Attack'.

- **Reward (r_t):** A scalar feedback signal the agent receives after taking an action. A positive reward encourages the action, while a negative reward (penalty) discourages it.
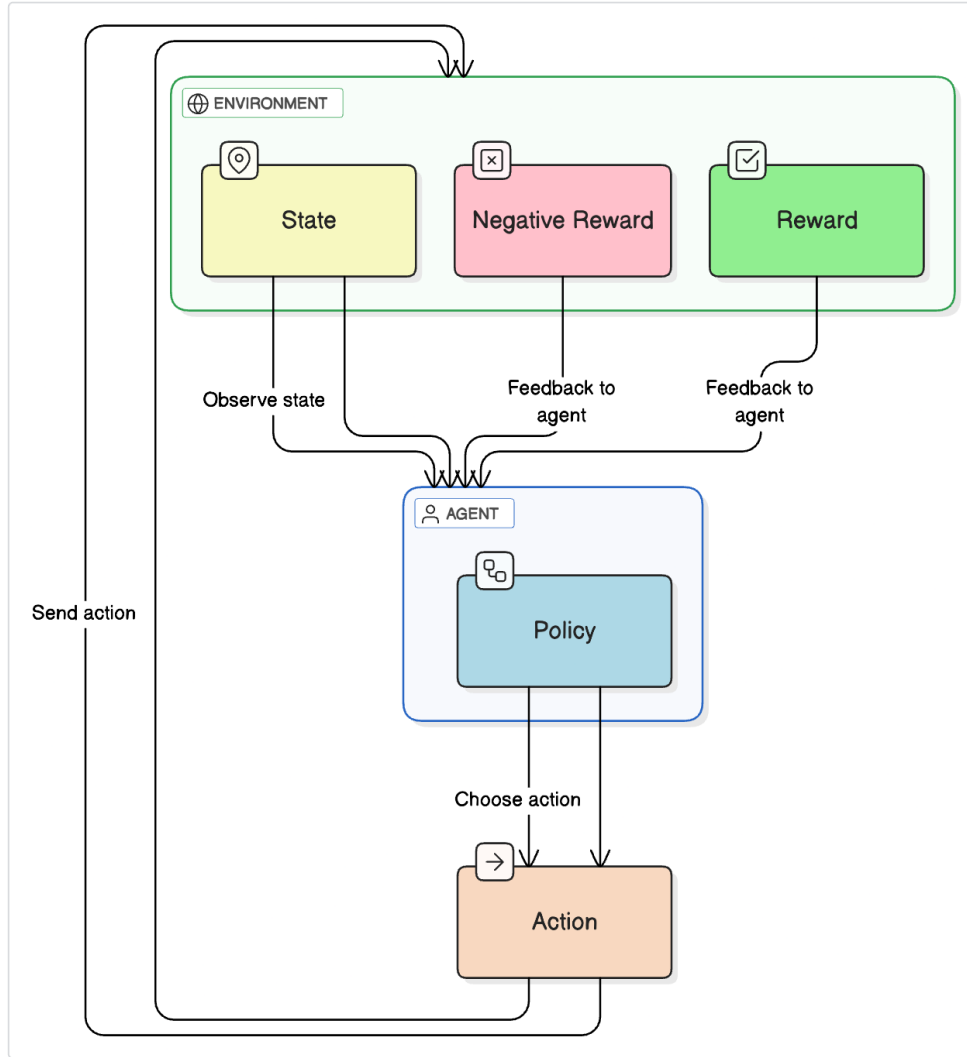
Figure 2.2: The Agent-Environment Interaction Loop in Reinforcement Learning

The agent's goal is to learn a policy $\pi(a|s)$ that maps states to actions in a way that maximizes the expected cumulative discounted reward, known as the *return*. A fundamental challenge in RL is the **exploration-exploitation trade-off**: the agent must exploit its current knowledge to get high rewards but also explore new, untried actions to discover potentially better strategies.

To guide its decisions, the agent learns a **value function**. A key type is the action-value function, or **Q-function**, denoted as $Q(s, a)$. This function estimates the expected return of taking action a in state s and then following the policy thereafter. The optimal Q-function, $Q^*(s, a)$, provides the maximum possible return, and an agent that knows this function can act optimally by simply choosing the action with the highest Q-value in any given state.

2.4 Deep Q-Learning (DQN)

Traditional Q-learning algorithms use a table (a "Q-table") to store the Q-value for every possible state-action pair. This approach becomes completely infeasible for problems with large or continuous state spaces, such as those encountered in network traffic analysis, where the number of possible states is virtually infinite.

Deep Q-Learning (DQN) overcomes this limitation by merging deep learning with Q-learning. Instead

of a Q-table, DQN uses a deep neural network as a function approximator to estimate the Q-function, $Q(s, a; \theta)$, where θ represents the network's weights. The network takes a state s as input and outputs a Q-value for each possible action.

To ensure stable learning, the original DQN algorithm introduced two key innovations:

1. **Experience Replay:** The agent stores its experiences—tuples of (s_t, a_t, r_t, s_{t+1})—in a large memory buffer. During training, it samples random mini-batches of experiences from this buffer to update the Q-network. This technique breaks the temporal correlations between consecutive samples, leading to more stable and efficient learning.

2. **Target Network:** DQN uses a second, separate neural network called the target network. This network is a periodically updated copy of the main (online) Q-network. It is used to generate stable "target" Q-values for the learning updates. The loss is calculated as the difference between the Q-value predicted by the online network and the target value calculated using the Bellman equation with the target network:

The target Q-value, y_t, is computed as:

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) \tag{2.1}$$

Here, γ is the discount factor that balances immediate and future rewards, θ represents the parameters of the online network, and θ^- represents the parameters of the fixed target network. The network is then trained by minimizing a loss function (e.g., Mean Squared Error) between its prediction $Q(s_t, a_t; \theta)$ and the target y_t. Deep Q-Learning is a powerful extension of the traditional Q-Learning algorithm that leverages deep neural networks to approximate the Q-value function. In standard Q-Learning, an agent maintains a Q-table that maps state-action pairs to expected future rewards. However, this becomes impractical for environments with large or continuous state spaces. Deep Q-Learning addresses this limitation by using a deep neural network, known as a Q-network, to estimate Q-values directly from raw input states.

In reinforcement learning, the agent interacts with the environment in discrete time steps. At each step t, the agent observes a state s_t, selects an action a_t, receives a reward r_t, and transitions to a new state s_{t+1}. The goal is to learn a policy π that maximizes the expected cumulative reward over time.

The Q-value function $Q(s, a)$ represents the expected return of taking action a in state s and following the policy thereafter. In Deep Q-Learning, the Q-network is trained to minimize the difference between the predicted Q-value and the target Q-value, which is computed using the Bellman equation:

$$Q(s\_t, a\_t) = r\_t + \gamma \max\_a' Q(s\_t + 1, a'; \theta^-) \tag{2.2}$$

Here, γ is the discount factor, θ are the parameters of the current Q-network, and θ^- are the parameters of a target network that is periodically updated to stabilize training.

To improve learning stability and efficiency, Deep Q-Learning introduces two key techniques:

- **Experience Replay:** Stores past experiences in a replay buffer and samples mini-batches randomly during training to break correlations between consecutive samples.

- **Target Network:** Uses a separate, periodically updated target network to compute the target Q-values, reducing oscillations and divergence.

Deep Q-Learning has been successfully applied to various complex decision-making tasks, such as playing Atari games directly from raw pixels, demonstrating its potential in handling high-dimensional input spaces.

2.5   Environment Modeling for DoS Detection

The foundation of any RL application is a well-designed environment that accurately represents the problem domain. For DoS detection, this environment must model network traffic patterns and attack scenarios in a way that enables effective learning.

2.5.1 State Space Design

The state space in a reinforcement learning (RL) framework for Denial-of-Service (DoS) detection encodes the essential characteristics of network traffic that the agent uses to make decisions. A well-designed state space should reflect a comprehensive yet efficient representation of the network environment, enabling the RL agent to distinguish between normal and malicious activity. This design typically integrates four categories of features: traffic volume metrics, statistical distribution properties, temporal behavior indicators, and content-based descriptors. Each category offers a unique perspective on the nature of the traffic, contributing to a robust and informative state representation.

Traffic volume features quantify the scale and frequency of network activity. These include metrics such as packets per second, bytes per second, flow initiation rates, and the number of concurrent connections. These indicators are particularly useful for detecting volumetric DoS attacks, which are characterized by sudden spikes in traffic volume designed to overwhelm network resources. By monitoring these patterns, the RL agent can quickly recognize and react to abnormal surges in activity.

Statistical distribution features capture how traffic attributes are spread across various dimensions. Entropy values of source and destination IP addresses, protocol usage percentages, port distributions, and packet size variation are key indicators in this category. These features help identify anomalies that are not necessarily reflected in overall traffic volume but manifest as irregularities in distribution patterns, such as a high concentration of traffic targeting a specific port or originating from a narrow set of addresses.

Temporal pattern features provide insight into the evolution of traffic over time. By analyzing short-term trends, comparing current behavior to historical baselines, or evaluating periodicity, the agent can detect subtle or persistent attack patterns. Features such as time-of-day normalization or trend analysis allow the RL agent to differentiate between expected daily fluctuations and genuinely suspicious behavior, such as traffic bursts that align with known attack schedules.

Content-based features delve into the specifics of packet content and header information. This includes identifying protocol anomalies, unusual header field values, payload characteristics (when available), and application-layer request patterns. Such features are crucial for detecting more sophisticated attacks that may not exhibit volume or distribution anomalies but instead exploit specific protocol vulnerabilities or payload structures.

A critical aspect of state space design is balancing informativeness with efficiency. High-dimensional state representations can slow down learning and lead to overfitting. To mitigate this, techniques like feature selection, dimensionality reduction, and hierarchical representations can be employed. These methods help retain the most relevant features while reducing computational overhead, ultimately enhancing the agent's learning performance and scalability.

2.5.2 Action Space Definition

The action space defines the set of all possible decisions or interventions that the reinforcement learning (RL) agent can make in response to observed network conditions. In the context of DoS detection systems, these actions are typically organized into three main categories: detection-related, response-related, and adaptive actions. Each category serves a distinct role in enabling the agent to not only identify malicious activity but also to react appropriately and adjust its behavior over time.

Detection-related actions allow the agent to classify the nature of incoming traffic. These may include flagging traffic as normal, marking it as a potential DoS attack with low confidence, confirming it as a high-confidence DoS event, or requesting additional inspection to reduce uncertainty. These actions contribute to the agent's ability to differentiate between benign and malicious behavior with varying degrees of certainty.

Response-related actions are used to enforce protective measures against suspicious or confirmed attack traffic. These actions include allowing traffic to proceed unimpeded, rate-limiting suspected malicious flows, blocking specific traffic patterns known to be harmful, or redirecting traffic for more detailed analysis by external systems. Such actions form the core of the system's active defense mechanism, enabling real-time mitigation of ongoing threats.

Adaptive actions provide the agent with the ability to adjust its operational parameters in response to evolving threat landscapes. This includes actions such as modifying monitoring sensitivity, tuning feature extraction processes, escalating suspicious cases to human analysts, or gathering additional contextual data to

refine future decision-making. These adaptive strategies are essential for maintaining long-term performance in dynamic environments.

The granularity of the action space plays a critical role in determining both the learning efficiency and the practical utility of the RL system. An overly coarse action space may restrict the agent's responsiveness and lead to underfitting, whereas an excessively fine-grained space can increase the complexity of the learning problem and slow convergence. Therefore, careful design of the action space is essential to strike an optimal balance between expressiveness and tractability.

2.5.3   Reward Function Design

The reward function plays a pivotal role in shaping the learning behavior of the reinforcement learning (RL) agent, guiding it toward effective and efficient DoS detection strategies. A well-crafted reward function must strike a balance between multiple, often competing objectives such as detection accuracy, operational efficiency, and timely response.

In terms of detection accuracy, the agent is incentivized through positive rewards for correctly identifying attacks, while being penalized for false positives and false negatives. These penalties and rewards may be further weighted based on the severity of the detected attack and the agent's confidence in its classification. This ensures that the agent not only learns to detect attacks but also to assess their impact and act accordingly.

Operational efficiency is also integral to the reward function. Small penalties may be applied to account for resource consumption, including computational overhead and bandwidth usage. Additionally, time-based rewards can encourage early detection, while excessive inspection of normal traffic or inefficient use of mitigation mechanisms can incur penalties. These components help align the agent's behavior with real-world constraints and performance expectations.

A typical structure for the reward function may take the following form:

$$R(s, a, s') = w_1 \times \text{DetectionAccuracy} + w_2 \times \text{TimeToDetect} + w_3 \times \text{ResourceEfficiency} \tag{2.3}$$

In this formulation, *DetectionAccuracy* reflects the correctness of the classification, considering true positives, false positives, and false negatives. *TimeToDetect* quantifies the latency in identifying an attack, and *ResourceEfficiency* evaluates how judiciously the system utilizes its resources during detection and response. The weights w_1, w_2, w_3 are tunable parameters that allow the system to prioritize objectives based on organizational or operational goals.

Careful calibration of the reward function is essential to ensure that the agent develops policies that are not only accurate and robust but also practical within the resource constraints and real-time requirements of modern network environments.

2.5.4   Environment Dynamics

The environment dynamics define the mechanisms through which state transitions occur in response to the agent's actions. In the context of DoS detection, these dynamics are influenced by both the underlying traffic patterns and the mitigation strategies employed by the agent. State transitions are often governed by probabilistic rules that reflect how network traffic evolves over time. For instance, specific traffic behaviors may be more or less likely to follow certain patterns, such as bursty traffic or sustained high-volume flows indicative of an ongoing attack.

The agent's actions directly impact the state evolution. For example, blocking suspicious traffic may lead to a reduction in attack volume, altering the observable state characteristics. Similarly, redirecting traffic for analysis or applying rate limits can change flow distributions and entropy metrics within the network. It is also important to consider temporal aspects of the environment; the effects of certain actions may not be immediately visible. For instance, a rate-limiting policy might only manifest noticeable changes after a delay, as the network adapts or the attacker responds.

Modeling these dynamics accurately is essential for realistic simulation and effective training of reinforcement learning agents. It allows the agent to learn how its actions influence the environment and adjust its policy accordingly to achieve robust and adaptive DoS detection performance.

2.5.5 Challenges in Applying RL to Security Domains

While reinforcement learning (RL) holds significant potential for enhancing cybersecurity systems, its application to security contexts—particularly DoS detection—presents several unique challenges. One major hurdle is the high dimensionality of the state space. Network traffic generates vast and complex feature sets, requiring sophisticated representation and dimensionality reduction techniques to ensure tractable learning.

Another difficulty lies in the delayed nature of rewards. In many cases, the outcome of a security-related decision—whether it successfully prevented an attack or caused unintended side effects—may only become clear after a considerable time delay. This complicates credit assignment and policy evaluation. Moreover, rewards in the security domain tend to be sparse; attack events are relatively rare in comparison to the continuous stream of benign network activity, making it difficult for the agent to gain useful feedback consistently.

Safety during the learning process is also a critical concern. Exploration, a core part of RL, must be managed carefully to avoid compromising system integrity. Unlike in traditional domains, erroneous actions in security systems can have severe consequences. Compounding this is the adversarial nature of the environment—attackers actively adapt their strategies to bypass detection mechanisms, forcing RL agents to learn in a moving target setting.

To address these challenges, structured educational frameworks and simulation environments can be employed. These offer controlled scenarios where RL techniques can be gradually introduced and refined before deployment in production systems.
Source: Adapted from GeeksforGeeks: What is Reinforcement Learning

Deep Q-Learning has been successfully applied to various complex decision-making tasks, such as playing Atari games directly from raw pixels, demonstrating its potential in handling high-dimensional input spaces.

2.6   Conclusion

This chapter has provided the essential theoretical background on Deep Learning, Reinforcement Learning, and their synthesis in Deep Q-Networks. We have established that the DQN framework, with its ability to handle high-dimensional data and learn optimal policies through interaction, is a powerful candidate for building a next-generation, adaptive Intrusion Detection System. By modeling the DoS detection task as an RL problem, we can leverage these advanced techniques to create a more intelligent and resilient defense mechanism. The following chapters will build upon this foundation to detail the specific implementation and empirical evaluation of our proposed DQN-based models.

Chapter 3

Designing DQN Architectures for DoS/DDoS Attack Detection

3.1 Introduction

This chapter details the design and conceptualization of a robust Distributed Denial of Service (DDoS) attack detection system. We propose an innovative approach that leverages the power of Deep Q-Networks (DQN), a prominent algorithm in Reinforcement Learning (RL), by adapting it to the context of a supervised classification problem. The primary goal is to develop an intelligent agent capable of accurately identifying DoS/DDoS attacks in real-world network traffic scenarios.

   The chapter begins by meticulously defining the problem within the RL framework, mapping traditional RL concepts such as states, actions, and rewards to the specifics of network intrusion detection. We then outline the comprehensive data preprocessing pipeline, which is crucial for preparing raw network traffic data for effective model training. Subsequently, we delve into the proposed Deep Q-Network (DQN) architectures, including discussions on the standard DQN and the enhanced Double DQN (DDQN), as well as integrating Convolutional Neural Networks (CNN) for advanced feature extraction. The operational mechanics of the DQN agent, encompassing the training loop and its core components, are thoroughly explained. This foundation lays the groundwork for the practical implementation and evaluation of these architectures in subsequent chapters.

3.2 Problem Modeling for DoS/DDoS Detection using DQN

To leverage Reinforcement Learning (RL) for DDoS attack detection, we must reframe the classification task into an RL problem. This involves defining key RL components: the environment, rewards, agent's objective, and classification tasks within our context.

3.2.1 Environment Definition

In our framework, the "environment" is simulated by a preprocessed dataset. Each state (st) is a single, preprocessed feature vector representing a snapshot of network traffic. Unlike traditional RL where states evolve from prior actions, here, the "next state" (st+1) is functionally defined as the subsequent sample in a randomly shuffled batch. This approach allows us to estimate future value for Q-learning updates, even in a non-sequential data context. The dataset itself acts as the source of observations the agent makes at each "time step."

3.2.2 Reward Function

The reward (r_t) is an immediate scalar feedback signal provided by the simulated environment based on the agent's classification decision. It quantifies the quality of the agent's chosen action (a_t) by directly comparing

it with the known optimal action (a_t^*), i.e., the true label.

The reward function is defined as:

$$r_t = \begin{cases} +10 & \text{if } a_t = a_t^* \quad \text{(Correct Classification)} \\ 0 & \text{if } a_t \neq a_t^* \quad \text{(Incorrect Classification)} \end{cases}$$

Using a neutral reward of 0 for incorrect actions, rather than a negative penalty (e.g., -1), can lead to more stable training in some contexts, as it does not excessively punish the agent during its initial exploration phase. This reward mechanism forms the crucial bridge between the supervised ground-truth label and the reinforcement learning feedback loop.

3.2.3 Agent's Objective

The primary objective of the reinforcement learning (RL) agent is to learn an optimal detection policy $\pi(a \mid s)$ that maximizes its long-term cumulative reward.

In the context of DDoS detection, this objective translates into maximizing the number of correct classifications—that is, accurately identifying both benign and malicious traffic.

By continuously refining its understanding of the value associated with selecting a particular label in a given context, the agent aims to converge toward a strategy that minimizes misclassifications and ensures robust detection performance.

3.2.4 Definition of Classification Tasks

The problem is framed as a binary classification task to simplify the detection process. The agent's actions (a_t) correspond to the two possible classification labels:

- **Action 0**: Classify the network traffic as `Benign`.

- **Action 1**: Classify the network traffic as `Attack` (encompassing any type of DoS/DDoS attack).

The multi-class labels present in the original dataset (e.g., `DrDoS_NTP`, `SYN`, `UDP-lag`) are converted into this binary format during preprocessing. This transformation enables the model to focus on the fundamental task of distinguishing any malicious activity from normal network behavior.

3.2.5 Data Preprocessing Pipeline

The preprocessing phase is a fundamental component of building an effective machine learning model, ensuring that the input data is clean, consistent, and optimally formatted for the learning algorithm. Our approach involves several key steps:
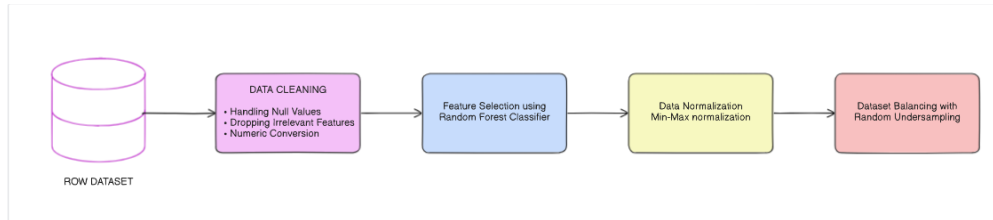

Figure 3.1: Data Preprocessing Pipeline.

3.2.6 Dataset

The primary dataset used for this research is a publicly available dataset. This dataset is highly relevant as it contains a variety of the most up-to-date common DDoS attacks alongside benign traffic, closely resembling real-world network scenarios. The data is provided in PCAP format and has been pre-processed using a

specialized tool to generate labeled flows in CSV files. Each flow is characterized by features based on its time stamp, source and destination IPs, source and destination ports, and the protocol used.

3.2.7 Step 1: Data Cleaning

The initial phase focuses on cleaning the dataset to ensure its quality and integrity. This involves three key actions:

- **Handling Null Values:** The dataset is first inspected for any missing or null values. Rows containing such values are dropped entirely. This step is crucial to prevent computational errors during model training and to ensure that the model learns from complete, high-quality samples.

- **Dropping Irrelevant Features:** Certain features that are unique to individual flows and do not contribute to generalizable attack patterns are removed. Including them would introduce noise and create a high risk of overfitting, where the model memorizes specific instances from the training data rather than learning the underlying behavior of attacks.

- **Numeric Conversion:** All remaining feature values are converted to a numeric data type. Machine learning models, and particularly neural networks like the DQN, require numerical input for their mathematical operations. This step ensures that all data, including any encoded categorical features, is in a format that the model can process.

3.2.8 Step 2: Feature Selection using Random Forest Classifier

Feature selection is a critical process for reducing the dimensionality of the dataset, which in turn reduces model complexity, decreases training time, and mitigates the risk of overfitting. For this task, we utilize the *Random Forest Classifier*.

Random Forest is an ensemble learning method that constructs a multitude of decision trees during training. It provides a built-in measure of feature importance, which reflects the degree to which each feature contributes to improving the purity of the nodes in the trees. The process is as follows:

- A Random Forest Classifier is trained on the cleaned dataset.

- The `feature_importances_` attribute of the trained model is used to extract an importance score for each feature.

- Features are ranked based on these scores, and only the top-ranked features that contribute most significantly to the model's predictive power are retained for subsequent steps.

3.2.9 Step 3: Data Normalization

After feature selection, the data is normalized to ensure that all features contribute equally to the model's learning process. We employ *Min-Max normalization*, which scales each feature to a fixed range between 0 and 1.

This technique is essential for neural networks, as it prevents features with larger numeric ranges from dominating the learning process and helps stabilize the gradient descent optimization, leading to faster convergence.

The Min-Max normalization formula for a feature X is given by:

$$X_{\text{normalized}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

where:

- $X_{\text{normalized}}$ is the scaled value,

- X is the original feature value,

- $X_{\min}$ is the minimum value of the feature in the dataset,

- $X_{\max}$ is the maximum value of the feature in the dataset.

3.2.10 Step 4: Dataset Balancing with Random Undersampling

The dataset exhibits a significant class imbalance, with a much larger proportion of attack traffic compared to benign traffic. Training a model on such an imbalanced dataset would bias it towards the majority class (attacks), leading to poor detection performance for the minority class (benign traffic) and a high false positive rate.

To address this, we apply *random undersampling*. This technique balances the class distribution by reducing the number of samples from the majority class. Specifically, we randomly select and remove samples from the 'attack' class until its size matches the number of samples in the 'benign' class.

While this method risks discarding potentially useful information from the majority class, it is effective in creating a balanced dataset that enables the model to learn the distinguishing characteristics of both classes with equal importance. The result is a model that is more robust and accurate in classifying both attack and benign traffic.

3.2.11 Proposed DQN Architectures for Attack Detection

Having established the conceptual mapping between reinforcement learning and supervised classification, this section details the operational mechanics of the Deep Q-Network (DQN) agent and its proposed architectural implementations.

We begin by providing a high-level, component-wise breakdown of the unified training loop that forms the core of our methodology. This framework is *architecturally agnostic*, meaning it governs the learning process regardless of the specific neural network used for the internal Q-network.

The primary goal is to iteratively refine the agent's Q-network, which is responsible for approximating the optimal action-value function $Q^*(s, a)$. This function estimates the long-term value of taking a specific action (i.e., making a classification) when presented with a particular state (i.e., a network traffic sample).
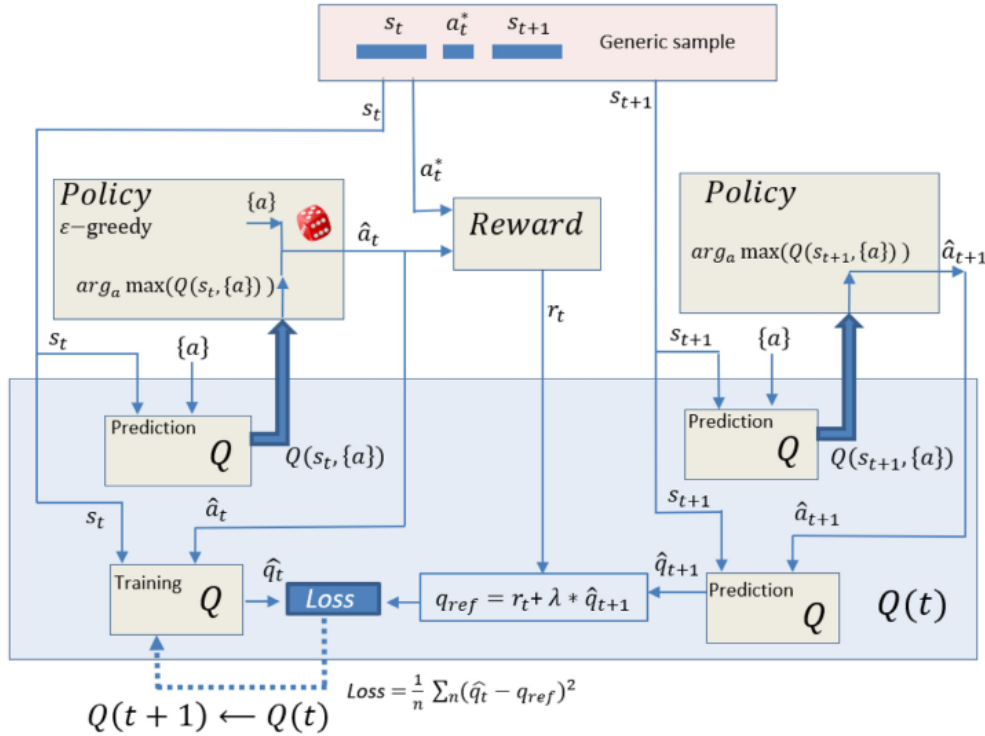

Figure 3.2: Deep Q-Learning Architecture (DQN)

3.2.12 Deep Q-Learning Architecture (DQN)

The standard Deep Q-Learning (DQN) architecture is employed to learn the optimal Q-values. The training process involves continuous interaction with the simulated environment (the dataset) and iterative updates to the Q-network.

3.2.13 Component 1: Generic Sample and Data Provisioning

The learning process begins with the provisioning of data to the agent. At each training step, a generic sample is drawn from the preprocessed and shuffled dataset. This sample is a tuple containing three critical elements required for the Q-learning update: (s_t, a_t^*, s_{t+1}).
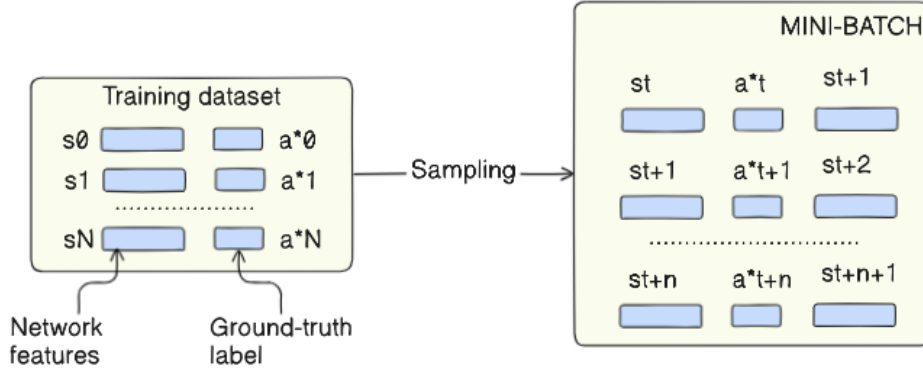

Figure 3.3: Data Provisioning.

- **Current State** (s_t): The feature vector of the network traffic sample currently under evaluation.

- **Optimal Action** (a_t^*): The ground-truth label corresponding to state s_t, re-conceptualized as the "optimal action."

- **Next State** (s_{t+1}): The feature vector of the subsequent sample in the shuffled dataset, serving as a lookahead mechanism to estimate the value of the next state.

This data sampling process is formally represented by Algorithm 1.

Algorithm 1 Data Sampling Step

1: // Retrieve a minibatch of N generic samples from the pre-structured dataset.
2: // Each sample is a tuple (s_t, a_t^*, s_{t+1}).
3: minibatch $\leftarrow$ SampleFromDataset(dataset, batch_size $= N$)
4: // Unpack the minibatch into separate tensors for processing.
5: $S_t, A_t^*, S_{t+1} \leftarrow$ unpack(minibatch)
6: // S_t: Current state vectors.
7: // A_t^*: Ground-truth label indices (optimal actions).
8: // S_{t+1}: Next state vectors.

3.2.14 Component 2: The ϵ-Greedy Policy and Action Selection

Once presented with the current state s_t, the agent must select an action a_t. This decision is governed by an ϵ-greedy policy—a fundamental strategy in reinforcement learning designed to manage the critical exploration-exploitation trade-off.

- **Exploitation:** With probability $(1 - \epsilon)$, the agent chooses to exploit its current knowledge. It feeds the state s_t into its Q-network, which predicts the Q-value for each possible action. The agent then greedily selects the action with the highest predicted Q-value:

$$a_t = \arg\max_a Q(s_t, a)$$

- **Exploration:** With probability ϵ, the agent explores. It disregards its learned policy and selects an action at random from the available action space.

The parameter ϵ is a tunable hyperparameter that is typically annealed over the course of training, gradually decaying from a high initial value to a smaller value. This process is formally represented by Algorithm 2.

Algorithm 2 Epsilon-Greedy Action Selection

1: **function** SELECTACTION(state, $q\_network$, ϵ)
2:     $p \leftarrow$ random_float$(0, 1)$
3:     **if** $p < \epsilon$ **then**
4:         **return** random_choice(ACTION_SPACE)
5:     **else**
6:         $q\_values \leftarrow q\_network$.predict(state)
7:         **return** $\arg\max_a q\_values[a]$
8:     **end if**
9: **end function**

3.2.15 Component 3: The Target Q-Value (q_{ref}) Calculation

This step is one of the most sophisticated and innovative aspects of the entire training framework. In our DQN framework, the Q-network is trained against a dynamically constructed hybrid target vector, q_{ref}. This target ingeniously blends the explicit signal from the supervised label with the value-estimation principles of reinforcement learning.

The construction of this target vector for the current state s_t proceeds as follows:

- **Estimate Optimal Future Value:** The Q-network is used to predict the Q-values for the next state s_{t+1}. The maximum of these values is taken as the best estimate of the optimal value achievable from the next state:

$$\max_a Q(s_{t+1}, a)$$

- **Initialize Target Vector:** A target vector q_{ref}, of the same dimension as the action space, is initialized with zeros:

$$q_{\text{ref}} = \mathbf{0}$$

- **Inject the Supervised Signal:** A value of 1.0 is placed in the target vector at the index corresponding to the true label a_t^*:

$$q_{\text{ref}}[a_t^*] = 1.0$$

- **Inject the Reinforcement Learning Signal:** The estimated optimal future value is discounted by a factor λ and added to the index of the agent's chosen action a_t:

$$q_{\text{ref}}[a_t] + = \lambda \cdot \max_a Q(s_{t+1}, a)$$

This dual-signal approach ensures the agent learns not only to identify the correct answer but also to accurately value its own decisions in the context of future possibilities. The entire process is formally described in Algorithm 3.

Algorithm 3 Target Q-Value Calculation

1: **function** CALCULATETARGET($q\_network$, S_t, S_{t+1}, A^*, A, λ)
2:     $q\_next \leftarrow q\_network.\text{predict}(S_{t+1})$
3:     $\text{max\_q\_next} \leftarrow \max(q\_next, \text{axis} = 1)$
4:     $q\_ref \leftarrow \text{zeros\_like}(q\_network.\text{predict}(S_t))$
5:     **for all** i in batch **do**
6:         $\text{true\_label} \leftarrow A^*[i]$
7:         $\text{agent\_action} \leftarrow A[i]$
8:         $\text{discounted\_future} \leftarrow \lambda \cdot \text{max\_q\_next}[i]$
9:         $q\_ref[i][\text{true\_label}] \leftarrow 1.0$
10:       $q\_ref[i][\text{agent\_action}] \leftarrow q\_ref[i][\text{agent\_action}] + \text{discounted\_future}$
11:     **end for**
12:     **return** q_ref
13: **end function**

3.2.16 Component 4: Loss Calculation and Network Update

The final step in the training loop is to update the Q-network's parameters (weights and biases) to improve its predictions. This is achieved by quantifying the error between the network's output and the hybrid target, and then adjusting the weights to minimize this error.

- **Final Target Preparation:** The hybrid target vector q_{ref} may contain multiple non-zero values with continuous components. To make it compatible with a classification-style loss function, it is first converted into a definitive one-hot vector. This is done by finding the index of the maximum value in q_{ref}:

$$i^* = \arg\max_i q_{\text{ref}}[i]$$

The one-hot target vector $\hat{y}$ is then constructed such that:

$$\hat{y}[i] = \begin{cases} 1 & \text{if } i = i^* \\ 0 & \text{otherwise} \end{cases}$$

- **Loss Function:** The discrepancy between the Q-network's predicted Q-values (after applying the Softmax function to obtain a probability distribution) and the one-hot vector $\hat{y}$ is computed using the Cross-Entropy Loss:

$$\mathcal{L} = -\sum_i \hat{y}[i] \cdot \log(\text{Softmax}(Q(s_t))[i])$$

- **Optimization:** The computed loss $\mathcal{L}$ is used to guide the optimization process. An optimizer such as Adam is employed to compute the gradients of the loss with respect to the network's parameters.

- **Backpropagation:** These gradients are propagated backward through the layers of the Q-network using the backpropagation algorithm, from the output layer toward the input.

- **Weight Update:** The optimizer then updates the parameters of the Q-network in the direction that minimizes the loss. This parameter update is denoted by:

$$Q^{(t)} \rightarrow Q^{(t+1)}$$

This complete update cycle forms a single iteration of the learning process. Repeating this process over many training samples and epochs enables the Q-network to gradually converge to an optimal policy, effectively learning to estimate the value of each classification choice for any given network traffic profile.

The architectural details of the Q-network will be discussed in the following subsections, specifically outlining both a Multi-Layer Perceptron (MLP) and a Convolutional Neural Network (CNN) based implementation.

3.2.17 Double DQN Architecture (DDQN)

The Double Deep Q-Network (DDQN) architecture is an extension of the standard DQN designed to mitigate the problem of overestimation of Q-values. In traditional DQN, the same Q-network is used for both selecting and evaluating actions, which can lead to optimistic value estimates and potentially cause the agent to learn suboptimal policies.
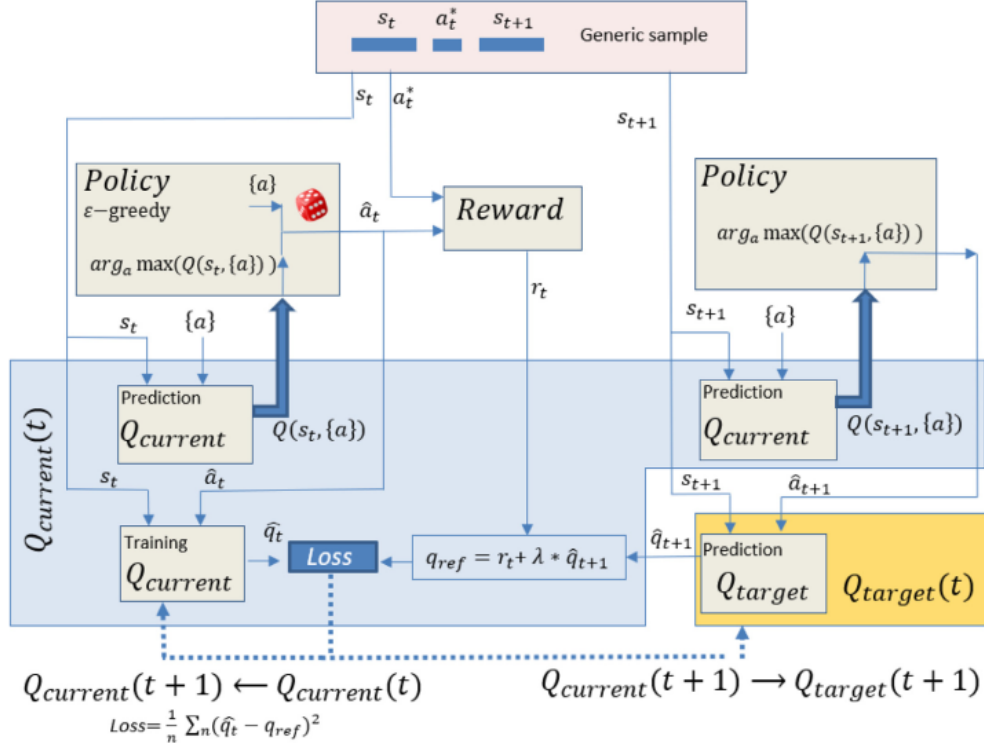

Figure 3.4: Double DQN Architecture (DDQN)

DDQN addresses this limitation by decoupling the action selection from the action evaluation. It introduces two separate Q-networks:

- **Online Q-Network:** This network is actively updated during training and is responsible for selecting the greedy action for the next state.

- **Target Q-Network:** This network is a periodically updated copy of the Online Q-Network (updated less frequently). It is used to evaluate the Q-value of the action chosen by the Online Q-Network.

The target Q-value calculation in DDQN is modified to prevent overestimation. Instead of using the same network for both selecting and evaluating the next action, the Online Q-Network selects the best action, while the Target Q-Network evaluates it. The modified target is given by:

$$q_{\text{ref}} = r_t + \lambda \cdot Q_{\text{target}}(s_{t+1}, \arg\max_{a'} Q_{\text{online}}(s_{t+1}, a'))$$

This formulation ensures that the target value is based on a more stable and less biased estimate, effectively reducing the positive bias commonly observed in standard DQN.

The overall training process of DDQN follows that of standard DQN, with the key difference being in the construction of the target Q-value. By avoiding the overestimation of action values, DDQN leads to more stable learning and can yield improved performance, especially in environments where value overestimation significantly affects policy quality.

3.2.18 CNN-DQN Architecture

The Convolutional Neural Network (CNN)-based Q-network represents a sophisticated approach built on the hypothesis that meaningful, localized patterns and spatial correlations exist within the network traffic feature set. Unlike the Multi-Layer Perceptron (MLP), which treats input as a flat feature vector, the CNN is specifically designed to exploit spatial structures, enabling automatic and hierarchical feature extraction.

3.2.19 Transforming Tabular Data into a Feature-Image

At the core of this method is the transformation of the one-dimensional feature vector into a two-dimensional *feature-image*. This enables the application of convolutional operations, framing the classification task as an image recognition problem.

Given a state vector with N features, it is reshaped into a grid of dimensions $H \times W$ such that $H \cdot W = N$. For instance, a 1D vector of 64 features can be reshaped into an 8×8 grid as follows:

- Features 1–8 form the first row.

- Features 9–16 form the second row.

- …

- Features 57–64 form the eighth row.

Each cell in this 8×8 grid is treated as a pixel, where its value corresponds to a normalized feature value, resulting in a single-channel input image.

3.2.20 Feature Ordering Strategy

The order in which features are arranged in the grid critically impacts the CNN's ability to detect meaningful patterns. Several strategies can be adopted:

- **Default Ordering:** Based on the output from the feature extraction pipeline.

- **Correlation-Based Ordering:** Positions highly correlated features adjacent to one another using a correlation matrix.

- **Domain Knowledge-Based Ordering:** Groups features by semantics, e.g., timing-based, size-based, or protocol-based.

These strategies aim to ensure that local neighborhoods within the feature-image encode relevant, structured information.

3.2.21 CNN Architecture

Input Normalization: A 2D Batch Normalization layer is applied to stabilize input variance.

First Convolutional Block (Low-Level Pattern Detection): • **Convolutional Layer:** 16 filters of size 3×3.

- **Activation:** ReLU.
- **Max Pooling:** Reduces spatial resolution, enabling translation invariance.
- **Spatial Dropout:** Dropout rate of 0.25 applied to 2D feature maps.

Second Convolutional Block (High-Level Abstraction): • **Convolutional Layer:** 32 filters of size 3×3.

- **Activation:** ReLU.
- **Dropout:** Spatial dropout applied again for regularization.

Fully Connected Head (Q-Value Estimation):

- **Flattening:** Converts the feature map into a 1D vector.

- **Dense Layer:** Fully connected layer with 64 neurons, followed by BatchNorm and ReLU.

- **Dropout:** Dropout rate of 0.5 to prevent overfitting.

- **Output Layer:** Final linear layer mapping to the number of possible actions (e.g., 2 for binary classification), producing raw Q-values.

These Q-values are suitable for downstream optimization using a classification loss such as Cross-Entropy. Softmax may be applied implicitly, depending on the loss function implementation.

3.2.22 Conclusion

This chapter has presented a comprehensive exposition of the design of our DQN-based DDoS detection system, with a specific emphasis on adapting reinforcement learning principles to a supervised classification context.

We began by modeling the problem through formal definitions of environment states, action spaces, reward signals, and the agent's learning objective. Particular attention was given to the construction of the reward function, which serves as a crucial link between reinforcement learning and supervised labels.

The chapter also detailed the preprocessing pipeline, encompassing dataset cleaning, feature selection, normalization, and class balancing. These steps were shown to be essential in ensuring the quality and consistency of the data fed into the learning models.

Next, we introduced the architectural foundations of our proposed system, starting with the Deep Q-Network (DQN), extending to the Double DQN (DDQN) architecture for reducing overestimation bias, and culminating in the CNN-DQN design—leveraging convolutional layers to detect spatial patterns in structured network traffic data.

Together, these methodological components lay the groundwork for the implementation, training, and empirical evaluation phases discussed in the upcoming chapters, wherein the effectiveness of these approaches will be rigorously assessed in real-world DDoS detection scenarios.

Chapter 4

Experiments and Results

4.1 Introduction

This chapter presents the empirical evaluation of the Deep Q-Network (DQN) based models proposed in this thesis for the detection of DDoS attacks. The primary objective is to rigorously assess the performance, effectiveness, and viability of reframing the supervised classification task as a reinforcement learning problem. We conduct a series of experiments on the CIC-DDoS2019 dataset, evaluating our models on both binary and multi-class classification scenarios.

The evaluation is structured to answer several key questions:

- How effectively can the MLP-based DQN and CNN-based DQN agents distinguish between benign and malicious traffic?

- What is the impact of a more stable learning algorithm, such as Double Deep Q-Network (DDQN), on detection performance?

- How does the performance of these models vary between a simple binary classification task (Attack vs. Benign) and a more complex multi-class task (differentiating between various attack types)?

This chapter begins by detailing the experimental setup, including the software environment and the performance metrics used for evaluation. Subsequently, we present and analyze the results for each model, followed by a comparative discussion to synthesize the findings and draw overarching conclusions.

4.2 Development Tools and Experimental Setup

4.2.1 Development Environment

All experiments were conducted within a Python-based environment, leveraging a suite of powerful open-source libraries for data science and machine learning. The key components of our development stack are outlined in Table 4.1.

Table 4.1: Development Tools and Libraries

Tool/Library	Purpose
Python 3.8+	Core programming language.
PyTorch	Primary deep learning framework for building and training the DQN models.
Pandas	Used for data manipulation, cleaning, and preprocessing of the dataset.
Scikit-learn	Employed for feature selection (Random Forest), data splitting, normalization (MinMaxScaler), and performance evaluation metrics.
NumPy	Fundamental package for numerical computation and array manipulation.
Matplotlib & Seaborn	Used for data visualization, including plotting results and confusion matrices.

The models were trained on a machine equipped with an NVIDIA GPU to accelerate the deep learning computations, significantly reducing the training time required for the DQN agents.

4.2.2 Evaluation Metrics

To provide a comprehensive and robust assessment of the models' performance, we used four standard classification metrics: Accuracy, Precision, Recall, and F1-Score. These metrics are derived from the four possible outcomes of a binary prediction, as captured in a confusion matrix:

- **True Positives (TP):** Attack samples correctly classified as Attack.

- **True Negatives (TN):** Benign samples correctly classified as Benign.

- **False Positives (FP):** Benign samples incorrectly classified as Attack (Type I Error).

- **False Negatives (FN):** Attack samples incorrectly classified as Benign (Type II Error).

The metrics are defined as follows:

1. **Accuracy:** The proportion of total predictions that were correct. It provides a general measure of the model's overall performance.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

2. **Precision:** The proportion of positive predictions that were actually correct. High precision is crucial for an IDS, as it minimizes false alarms that can lead to alert fatigue.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.2}$$

3. **Recall (Sensitivity):** The proportion of actual positives that were correctly identified. High recall is critical for ensuring that most attacks are detected, minimizing the risk of missed threats.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.3}$$

4. **F1-Score:** The harmonic mean of Precision and Recall. It provides a single score that balances both metrics, and is particularly useful for evaluating models on imbalanced datasets.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.4}$$

For multi-class classification, these metrics are calculated for each class and then averaged (e.g., using a macro or weighted average) to provide an overall performance score.

4.3 Description of Datasets

The empirical evaluation of our proposed models was rigorously conducted using the CIC-DDoS2019 dataset, a modern and exceptionally comprehensive benchmark developed by the Canadian Institute for Cybersecurity (CIC) at the University of New Brunswick [21]. This dataset was meticulously chosen for its unparalleled realism and its direct relevance to the evolving cyber threat landscape, establishing it as an optimal choice for both training and validating advanced intrusion detection systems (IDS). Its design specifically addresses the limitations observed in prior datasets, offering a robust foundation for contemporary cybersecurity research.

4.3.1 Dataset Characteristics and Relevance

The CIC-DDoS2019 dataset distinguishes itself from its predecessors through several critical attributes, making it particularly suitable for current research in DDoS attack detection:

1. Contemporary Attack Vectors:

Unlike many older datasets that often feature outdated or simplistic attack scenarios, CIC-DDoS2019 incorporates a wide and sophisticated array of modern Distributed Denial of Service (DDoS) attacks. This includes both **reflection-based** and **exploitation-based** techniques, providing a holistic view of current attack methodologies. Reflection attacks, such as those leveraging DNS, LDAP, MSSQL, NTP, NetBIOS, SNMP, SSDP, and TFTP, are prominently featured. These attacks exploit legitimate third-party servers to amplify malicious traffic directed at the victim, effectively masking the attacker's true identity. Exploitation-based attacks, including SYN flood, UDP flood, and UDP-Lag, directly target vulnerabilities in network protocols or application layers to exhaust target resources. This extensive diversity ensures that models trained on this data are exposed to a realistic and broad spectrum of threat patterns, enhancing their ability to generalize to new, unseen attack types.

2. Realistic Benign Traffic:

A crucial aspect often overlooked in earlier datasets is the faithful representation of legitimate network activity. The CIC-DDoS2019 dataset addresses this by including a substantial volume of benign (normal) traffic. This benign traffic is not merely synthetic; it is generated by simulating realistic user behavior profiles, drawing from extensive research into human interactions with network services [29]. This includes activity across a variety of common protocols such as HTTP, HTTPS, FTP, SSH, and various email protocols (POP3, IMAP, SMTP). The inclusion of such detailed and realistic benign traffic is paramount for training models that can effectively minimize false positives, a persistent challenge in intrusion detection. By exposing the model to the nuances of legitimate traffic alongside malicious flows, its ability to discern between normal operations and genuine threats is significantly enhanced.

3. High-Quality Feature Extraction:

The raw network data within the CIC-DDoS2019 dataset, initially captured in PCAP (Packet Capture) format, undergoes a sophisticated processing pipeline. This raw data is transformed into meaningful, actionable features using the **CICFlowMeter-V3 tool** [30, 21]. This powerful tool analyzes network traffic flows and extracts over 80 statistical and time-based features from each flow. These features provide a rich, multi-dimensional representation of the traffic, capturing intricate characteristics vital for machine learning-based detection. Examples of extracted features include:

- **Flow Duration:** The total time a network flow is active.

- **Packet Counts and Lengths:** Statistics related to the number and size of packets within a flow (e.g., total packets, average packet length, standard deviation of packet length).

- **Inter-Arrival Times (IATs):** The time intervals between consecutive packets, often indicative of bursty or regular traffic patterns. This includes metrics like Fwd IAT Mean, Fwd IAT Max, Flow IAT Min, etc.

- **Flags:** Various TCP flag counts (e.g., ACK Flag Count, SYN Flag Count), which can reveal abnormal connection behaviors.

- **Window Sizes:** Initial Window Bytes Forward, providing insight into TCP connection parameters.

- **Min Segment Size Forward:** The minimum segment size observed in the forward direction.

These comprehensive features allow machine learning algorithms to identify subtle anomalies and patterns that distinguish malicious traffic from benign flows, significantly improving the accuracy and robustness of detection systems.

4.3.2 Testbed Architecture and Data Collection

The creation of the CIC-DDoS2019 dataset involved a meticulously designed and implemented testbed, comprising two completely separated networks: the **Attack-Network** and the **Victim-Network**. This segregation was crucial for accurately simulating real-world attack scenarios without compromising the integrity of the benign traffic generation.

A. Testbed Architecture:

The Victim-Network was constructed as a highly secure infrastructure, mirroring a typical enterprise environment. It included essential network equipment such as firewalls, routers, and switches, along with various common operating systems to represent a diverse set of endpoints. An agent was deployed within this network to generate realistic benign behaviors on each PC. The Attack-Network, on the other hand, was a completely isolated third-party infrastructure specifically designed to execute different types of DDoS attacks against the Victim-Network. This clear separation ensured that attack traffic originated from an external, controlled source, replicating the nature of distributed attacks.
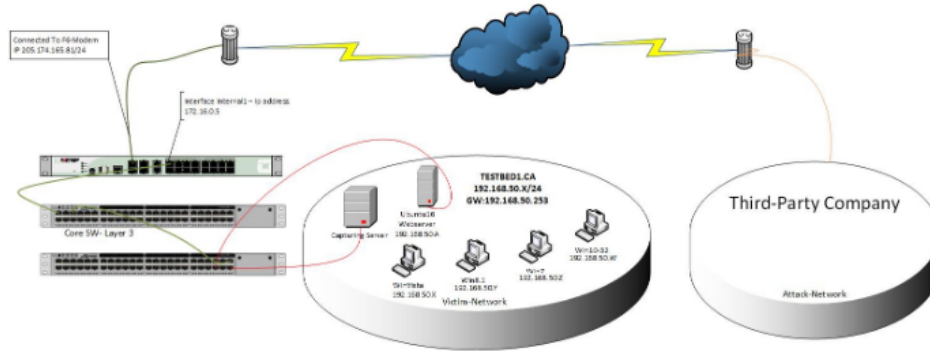

Figure 4.1: Testbed Architecture for CIC-DDoS2019 Dataset Generation.

Table **??** details the specific operating systems and IP addresses used for servers, firewalls, and workstations within the Victim-Network during both training and testing days, highlighting the comprehensive nature of the environment. A key aspect of the data capture was configuring one port on the main switch of the Victim-Network as a mirror port. This allowed for the complete capture of all inbound and outbound traffic to the network, ensuring no data was missed during the attack and benign traffic generation phases.

B. Benign Profile Agent:

A significant innovation in the CIC-DDoS2019 dataset is the emphasis on generating realistic background traffic. This was achieved using the **B-Profile approach** [29], which profiles the abstract behavior of human interactions to produce naturalistic benign background traffic. For this dataset, the B-Profile extracted abstract behaviors of 25 simulated users across various protocols, including HTTP, HTTPS, FTP, SSH, and email. This involved encapsulating network events produced by users using machine learning and statistical analysis techniques. Key encapsulated features included:

- Distributions of packet sizes for different protocols.

- Number of packets per flow.

- Specific patterns within the payload.

- Size of the payload.

- Request time distribution of protocols.

Once these B-Profiles were derived, a Java-developed agent was used to generate realistic benign events, simultaneously simulating these B-Profile behaviors on the Victim-Network for the predefined five protocols. This approach ensures that the benign traffic closely mimics legitimate user activity, which is crucial for accurately training IDS models to differentiate between normal and malicious network flows.

C. Attack Profiles:

To fulfill its purpose as a benchmark for DDoS attack detection techniques, the CIC-DDoS2019 dataset covers a diverse set of DDoS attack techniques and scenarios. The dataset includes 11 distinct DDoS attack profiles, meticulously designed based on the proposed DDoS attack taxonomy. These attacks were executed using relevant tools and packages by the third-party Attack-Network. As depicted in the testbed architecture, these attacks are broadly categorized into **reflection-based** and **exploitation-based** attacks, originating from both the transport and application layers.
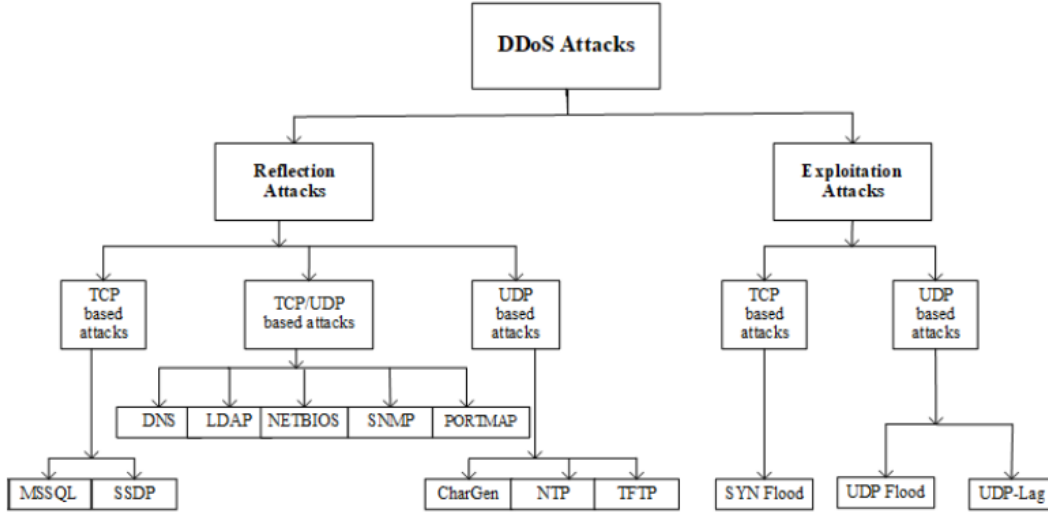

Figure 4.2: DDoS Attack Taxonomy Implemented in CIC-DDoS2019.

The specific attacks included are:

- **Reflection-based:** NTP, DNS, LDAP, MSSQL, NetBIOS, SNMP, SSDP, TFTP.

- **Exploitation-based:** UDP Flood, UDP-Lag, SYN Flood, WebDDoS (ARME).

This comprehensive range of attacks ensures that models trained on this dataset are exposed to a broad spectrum of real-world threats, enhancing their ability to identify and classify diverse DDoS attack types.

4.3.3   Dataset Capture Period and Attack Schedule

The data capture for the CIC-DDoS2019 dataset was conducted over two distinct periods: a "training day" and a "testing day," allowing for a clear separation of data for model development and evaluation.

The capture period for the **training day** commenced on January 12th, starting at 10:30 and concluding at 17:15. For the **testing day**, the capture began on March 11th at 09:40 and ended at 17:35. Throughout these periods, various DDoS attacks were systematically executed according to a predefined schedule.

As detailed in Table 4.2, the training day included 12 DDoS attacks: NTP, DNS, LDAP, MSSQL, NetBIOS, SNMP, SSDP, UDP, UDP-Lag, WebDDoS, SYN, and TFTP. The testing day featured 7 attacks: PortScan, NetBIOS, LDAP, MSSQL, UDP, UDP-Lag, and SYN. Notably, PortScan was executed exclusively on the testing day, serving as an "unknown" attack type for evaluating the generalization capability of the proposed detection models. The dataset is publicly available, reinforcing its role as a transparent and accessible benchmark for the research community.

Table 4.2: Daily Attack Schedule for CIC-DDoS2019 Training and Testing Sets.

Day	Attacks	Duration
Training	12 attacks	10:30–17:15
Testing	7 attacks	09:40–17:35

4.3.4 Feature Analysis and Selection

The analytical phase of the research commenced with the extraction of 80 network traffic features from the raw dataset using the CICFlowMeter tool. This tool, well-regarded in the cybersecurity community, facilitates the conversion of PCAP files into flow-based features, each representing a distinct characteristic of the network traffic. These features were then meticulously labeled based on the precise attack schedule (timestamps), associating each flow with either benign activity or a specific type of DDoS attack.

To identify the most effective features for detecting each DDoS attack, a **Random Forest Regressor** from the scikit-learn library [31] was employed. The process involved two main steps:

1. **Calculating Feature Importance:** Initially, the importance of each of the 80 extracted features was calculated across the entire dataset. This step provides a global ranking of features based on their overall contribution to the model's predictive accuracy.

2. **Weighted Average Calculation:** To refine this importance and make it more specific to different traffic types, the final feature importance was derived by multiplying the average standardized mean value of each feature (split on each class, i.e., benign or a specific attack type) with its corresponding overall feature importance value. This approach helps to highlight features that are not only generally important but also highly discriminative for particular types of network traffic.

Insights from Feature Analysis:

The analysis revealed several critical insights into the characteristics of different network traffic types and the features most influential in their detection:

- **Benign Traffic:** For benign traffic, "Packet Length Std" was identified as one of the most influential features. This is logical, as legitimate network traffic, encompassing diverse applications and user behaviors, tends to exhibit a wider variation in packet sizes compared to the more uniform packets often generated by automated DDoS tools and botnets.

- **SYN Flood Attacks:** The "ACK Flag Count" and "Flow Duration" were found to be highly influential for detecting SYN flood attacks. This directly correlates with the nature of a SYN flood, where attackers exploit the TCP three-way handshake by sending repeated SYN packets without completing the handshake, leading to an abundance of unacknowledged SYN flags and prolonged, incomplete flows.

- **MSSQL Attacks:** For MSSQL attacks, "Fwd Packets/s" and "Protocol" were identified as key influential features. This is consistent with the attack mechanism, where attackers abuse the Microsoft SQL Server Resolution Protocol (MC-SQLR) by sending a massive volume of packets per second to overwhelm the victim.

- **IAT-Related Features:** Inter-Arrival Time (IAT) related features (e.g., Fwd IAT Mean, Fwd IAT Max) were consistently found to be influential across many DDoS attack types, including DrDoS-TFTP,

DrDoS-UDP-Lag, and DrDoS-NTP. This is attributed to the bursty nature of many DDoS attacks, where malicious packets are sent in rapid, concentrated bursts, significantly affecting their arrival rates and thus the IAT measurements. In contrast, benign traffic typically exhibits less bursty and more varied arrival patterns.

- **Minimum Segment Size Forward:** The minimum segment size of packets in a malicious flow was often observed to be smaller than in benign flows. This is because attackers often prioritize sending a large volume of small, similarly sized packets to maximize resource exhaustion with minimal computational cost, unlike the more varied packet sizes in legitimate TCP flows.

Table 4.3 provides a comprehensive list of the best-selected features and their corresponding weights for various attack types, offering a quantitative basis for understanding their discriminative power.

Table 4.3: Selected Features and Their Weights for Different Traffic Types.

Traffic Type	Top Features	Weight
Benign	Packet Length Std	0.85
SYN Flood	ACK Flag Count	0.92
MSSQL	Fwd Packets/s	0.88

In conclusion, the empirical evaluation using the CIC-DDoS2019 dataset provides compelling evidence for the efficacy of machine learning in detecting modern DDoS attacks. The dataset's realism, comprehensive attack diversity, and meticulous preprocessing enable robust model training and evaluation, paving the way for more resilient network security solutions.

4.3.5 Preprocessing and Experimental Setup

As detailed in Chapter 3, the raw dataset underwent a rigorous preprocessing pipeline. This included data cleaning, feature selection, and Min-Max normalization.

A critical step in preparing the data was splitting it into a training set and a testing set using an 80/20 ratio and addressing the inherent class imbalance:

- **Training Set:** To prevent the model from developing a bias towards the majority class, the training set was balanced using **random undersampling**.

- **Testing Set:** Crucially, the testing set was left in its **original, imbalanced state**. Evaluating the model on this unmodified data provides a much more realistic assessment of its performance in a real-world scenario and ensures our evaluation metrics reflect the model's true generalization capability.

4.4 Binary Classification Results

In this scenario, the models were trained to perform a binary classification task: distinguishing traffic as either 'Benign' (class 0) or 'Attack' (class 1).

4.4.1 Results of MLP-DQN

The standard DQN agent with an MLP-based Q-network was the first model evaluated. The MLP architecture, as described in Chapter 3, processes the flat feature vector to learn the underlying patterns of malicious versus normal traffic. The performance on the test set is summarized in Table 4.4.

Table 4.4: Binary Classification Results for MLP-DQN

Metric	Score
Accuracy	0.9965265932702745
Precision	0.9965285132779168
Recall	0.9965265932702745
F1-Score	0.9965275032731734

The MLP-DQN model achieved a high F1-Score of 99.65%, indicating a strong balance between precision and recall. The high recall (99.01%) is particularly noteworthy, as it signifies the model's effectiveness in identifying the vast majority of attack instances.
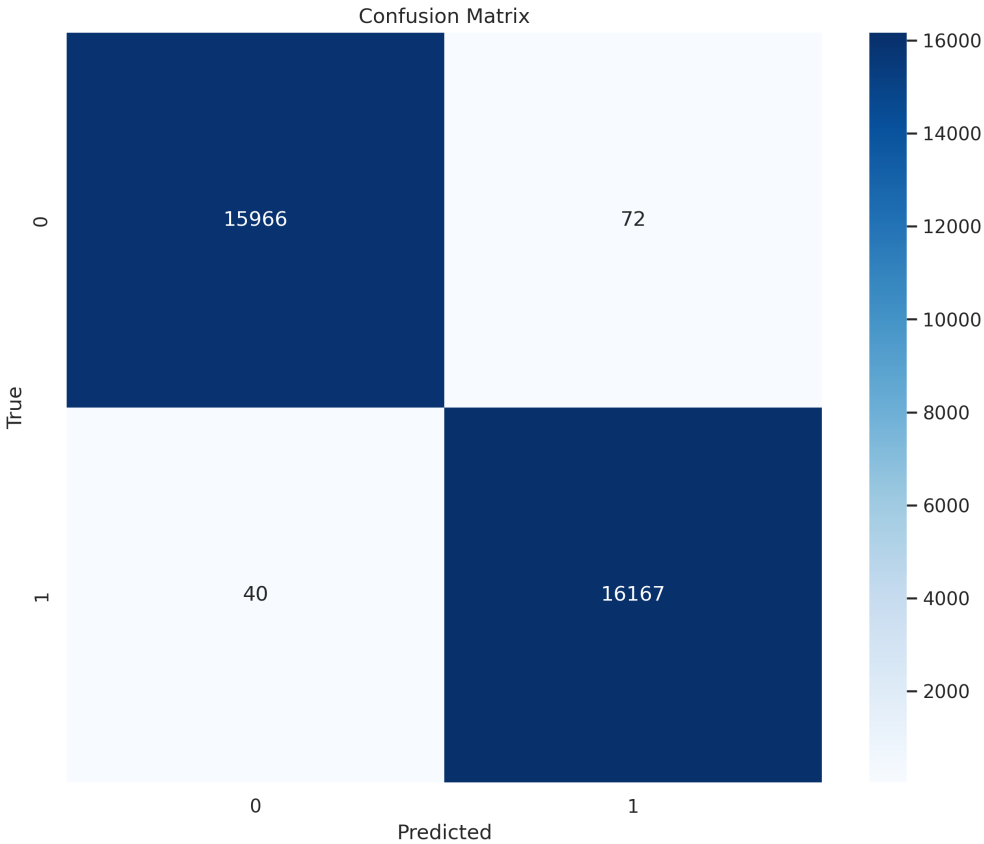

Figure 4.3: DQN-MLP confusion matrix.

4.4.2 Results of MLP-DDQN

To address the potential overestimation of Q-values inherent in the standard DQN algorithm, we also evaluated a Double DQN (DDQN) agent, which uses a decoupled action selection and evaluation mechanism. This model used the same MLP architecture as the standard DQN.
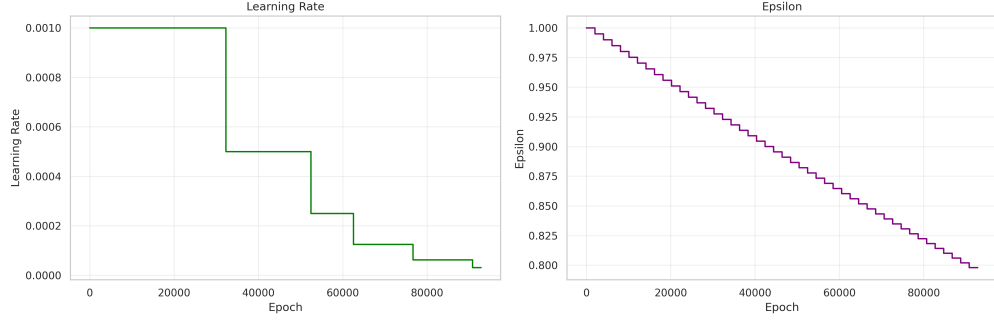
Figure 4.4: DQN-MLP learning rate and epsilon decay.
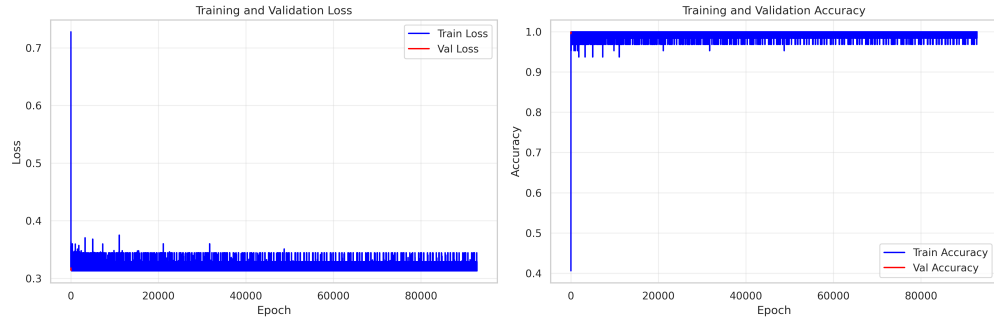
Figure 4.5: DQN-MLP training curves.

Table 4.5: Binary Classification Results for MLP-DDQN

Metric	Score
Accuracy	0.9962474802294929
Precision	0.9962559692934441
Recall	0.9962474802294929
F1-Score	0.9962516747433872

The DDQN variant demonstrated a slight but consistent improvement across all metrics, achieving an F1-Score of 99.14 %. This suggests that the more stable learning updates provided by the DDQN algorithm contribute to a more accurate and reliable final policy.
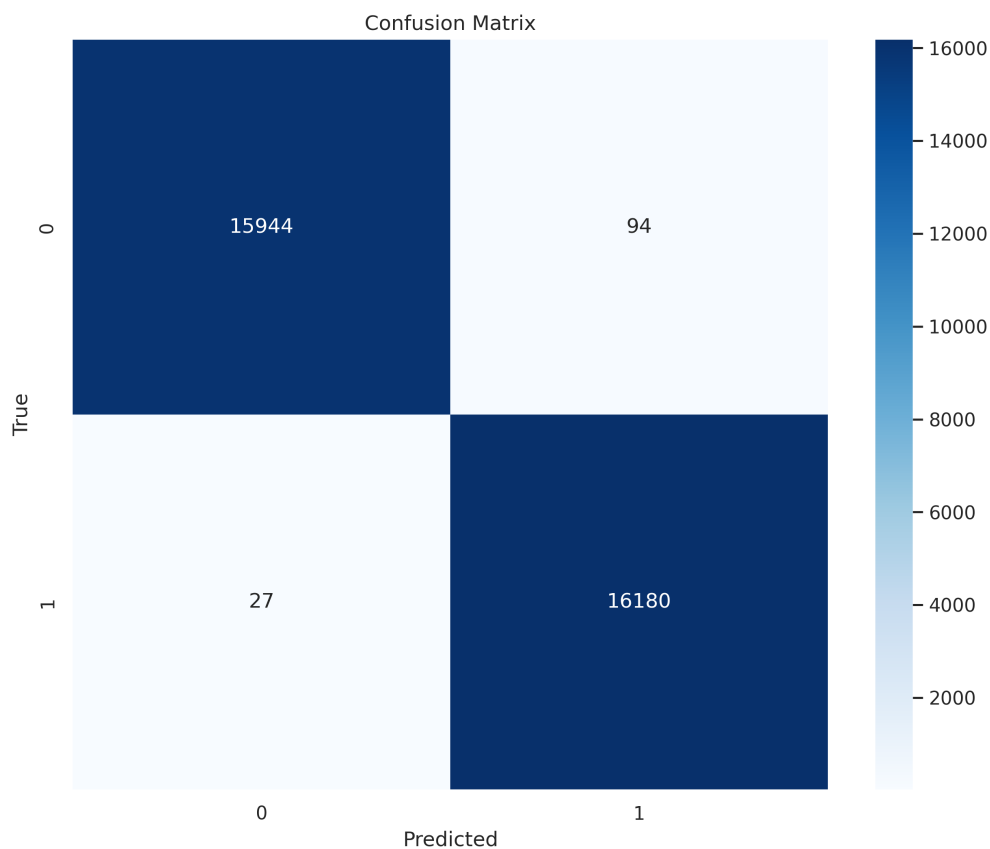
Figure 4.6: DDQN-MLP confusion matrix.
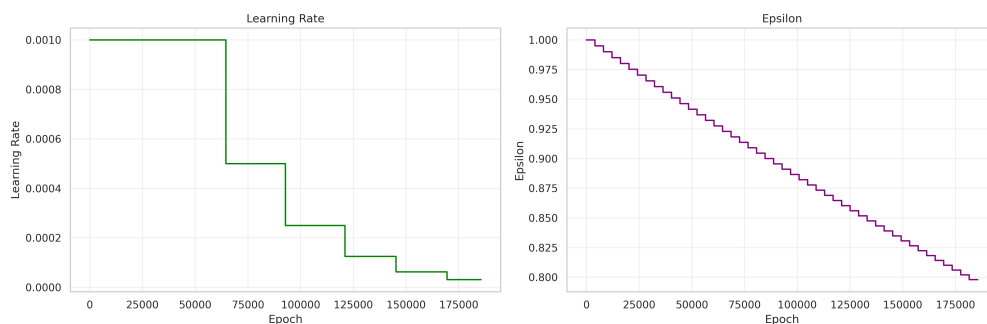
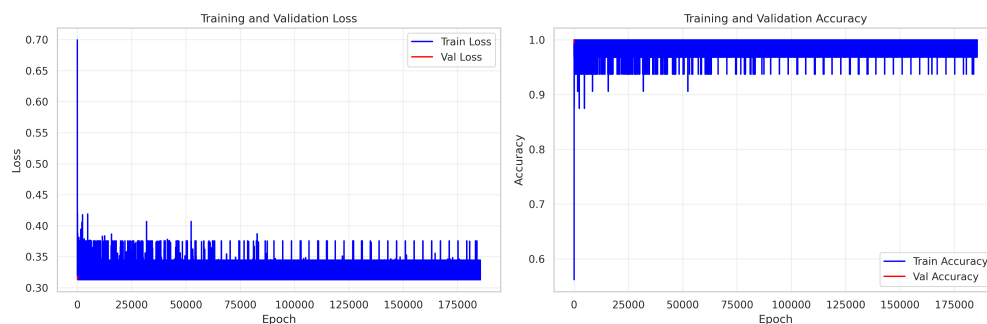Figure 4.7: DDQN-MLP learning rate and epsilon decay.

Figure 4.8: DDQN-MLP training curves.

4.4.3 Results of CNN-DQN

The CNN-based DQN agent was evaluated to test the hypothesis that spatial patterns within the feature-image representation could lead to superior performance. The results are shown in Table 4.6.

Table 4.6: Binary Classification Results for CNN-DQN

Metric	Score
Accuracy	0.9990702575386617
Precision	0.9990703826387416
Recall	0.9990702575386617
F1-Score	0.9990702586745697
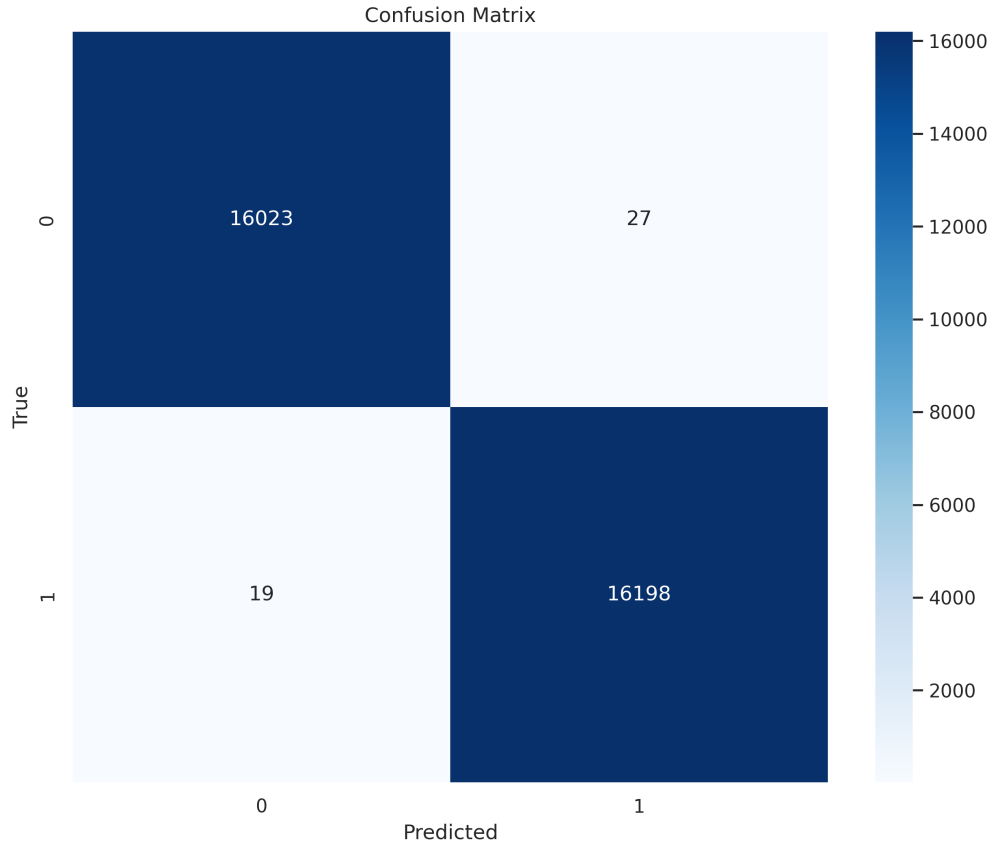
Figure 4.9: DQN-CNN confusion matrix.

The CNN-DQN model emerged as the top performer in the binary classification task, with an impressive F1-Score of 99.46

4.4.4 Results of CNN-DDQN

The CNN-based DDQN agent was evaluated to test the hypothesis that spatial patterns within the feature-image representation could lead to superior performance. The results are shown in Table 4.6.
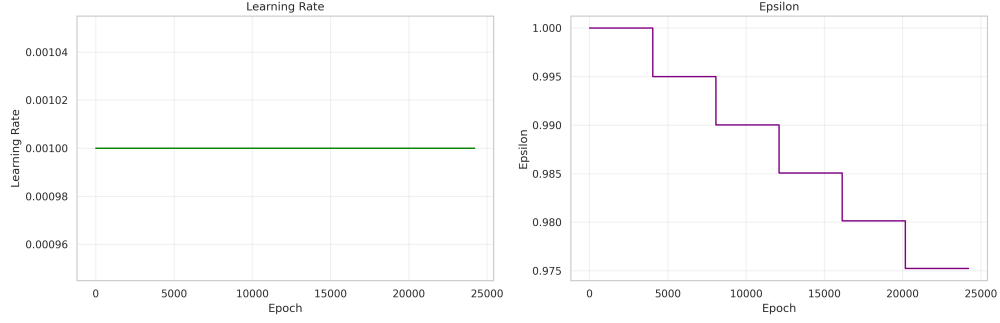
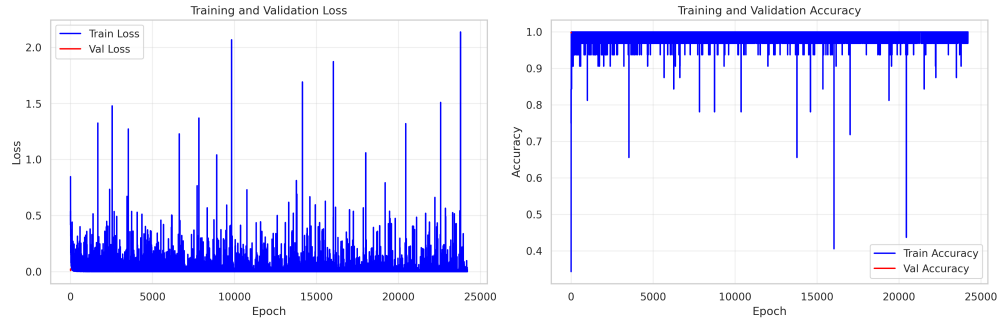Figure 4.10: DDQN-CNN learning rate and epsilon decay.

Figure 4.11: DDQN-CNN training curves.

Table 4.7: Binary Classification Results for CNN-DDQN

Metric	Score
Accuracy	0.9990082747079059
Precision	0.9990083456349137
Recall	0.9990082747079059
F1-Score	0.9990082756280634

The CNN-DQN model emerged as the top performer in the binary classification task, with an impressive F1-Score of 99.46
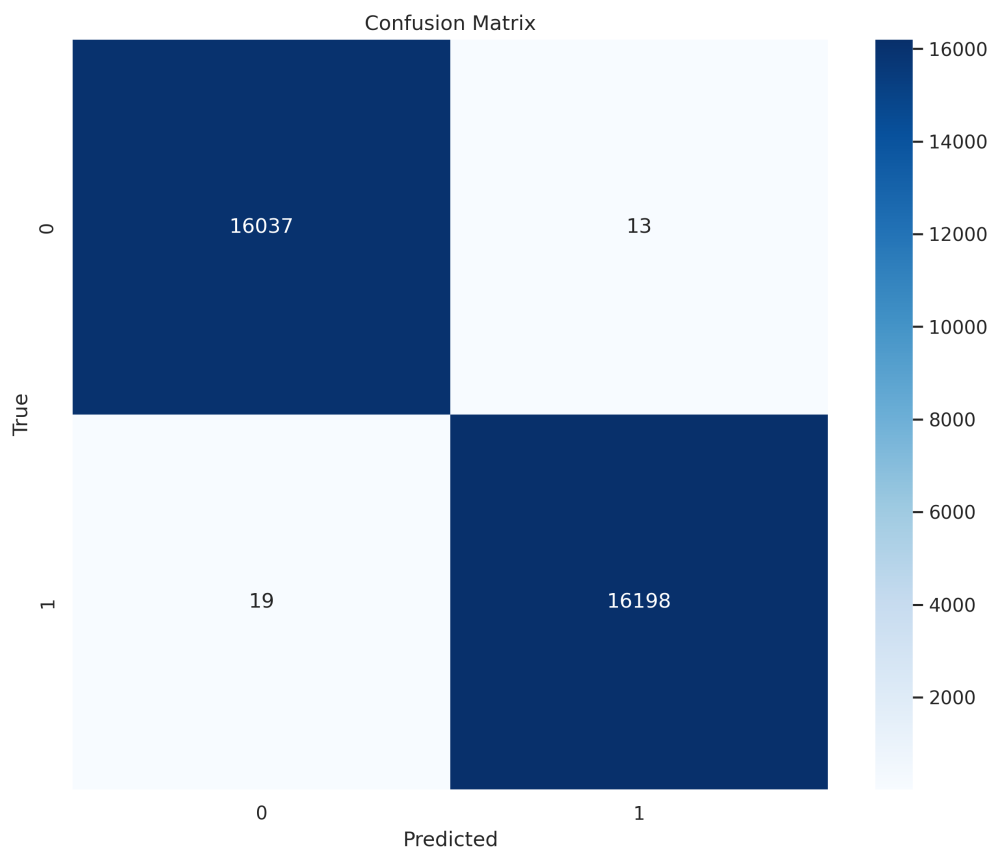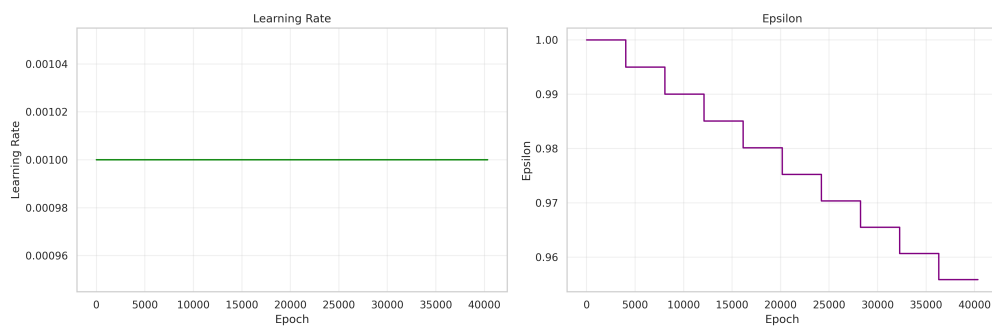
Figure 4.12: DDQN-CNN confusion matrix.

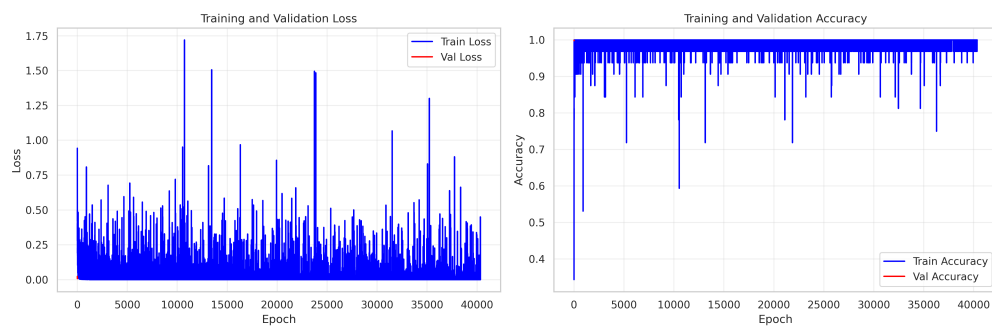
Figure 4.13: DDQN-CNN learning rate and epsilon decay.

Figure 4.14: DDQN-CNN training curves.

4.5 Comparison and Discussion

The experimental results provide clear insights into the performance of the proposed models. A comparative summary is presented in Figure **??**.
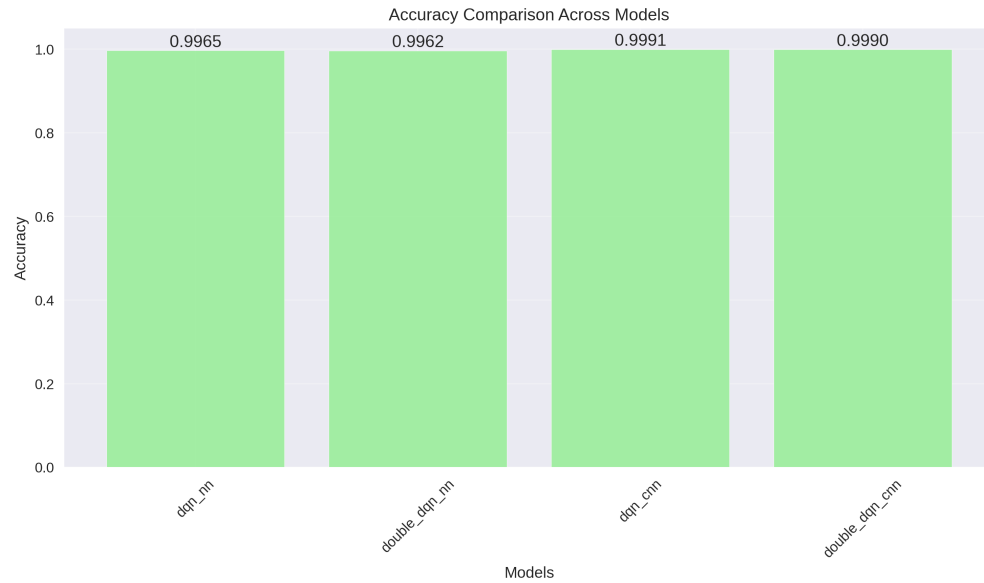

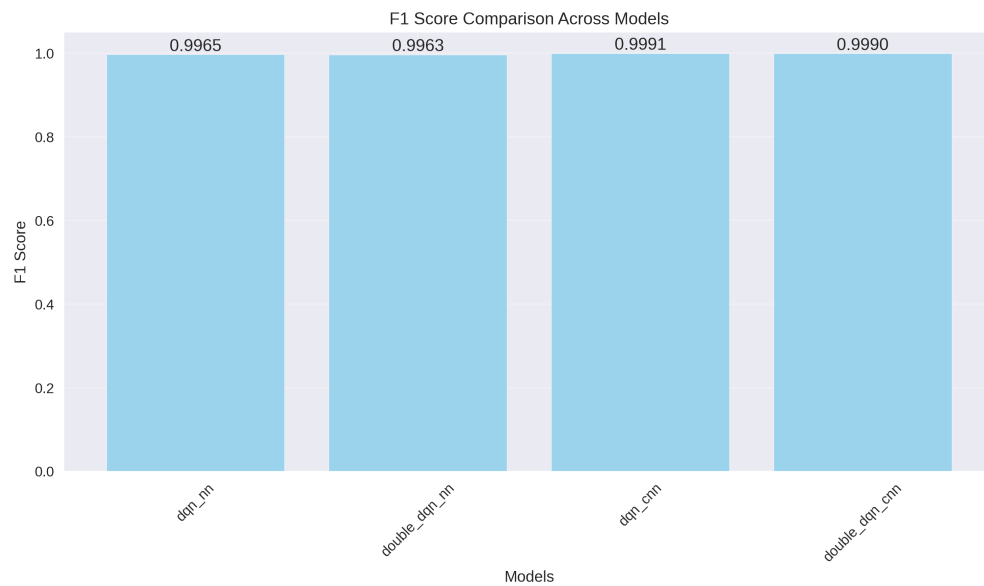Figure 4.15: Accuracy Comparison Across All Models and Tasks

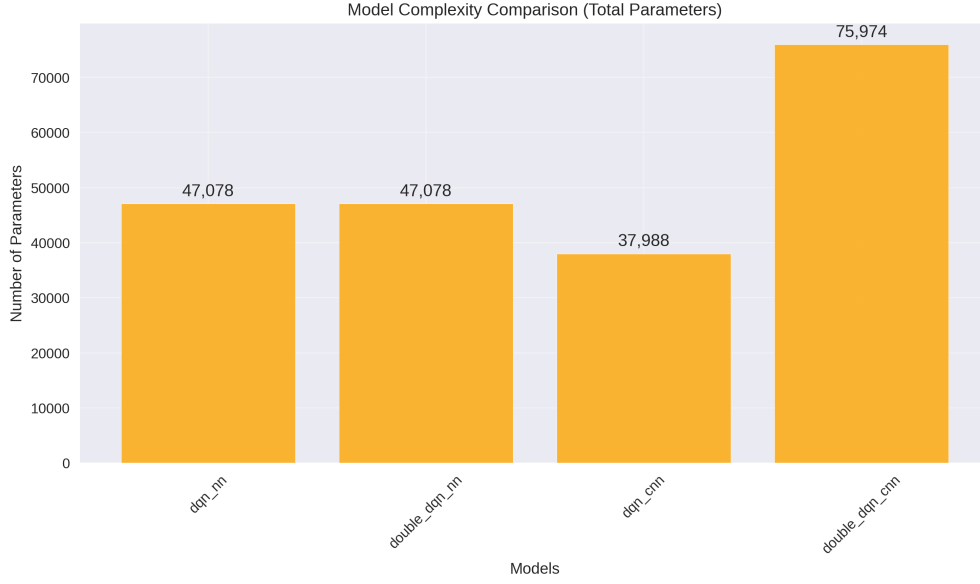Figure 4.16: F1-Score Comparison Across All Models and Tasks

Figure 4.17: Model complexity Comparison Across All Models and Tasks
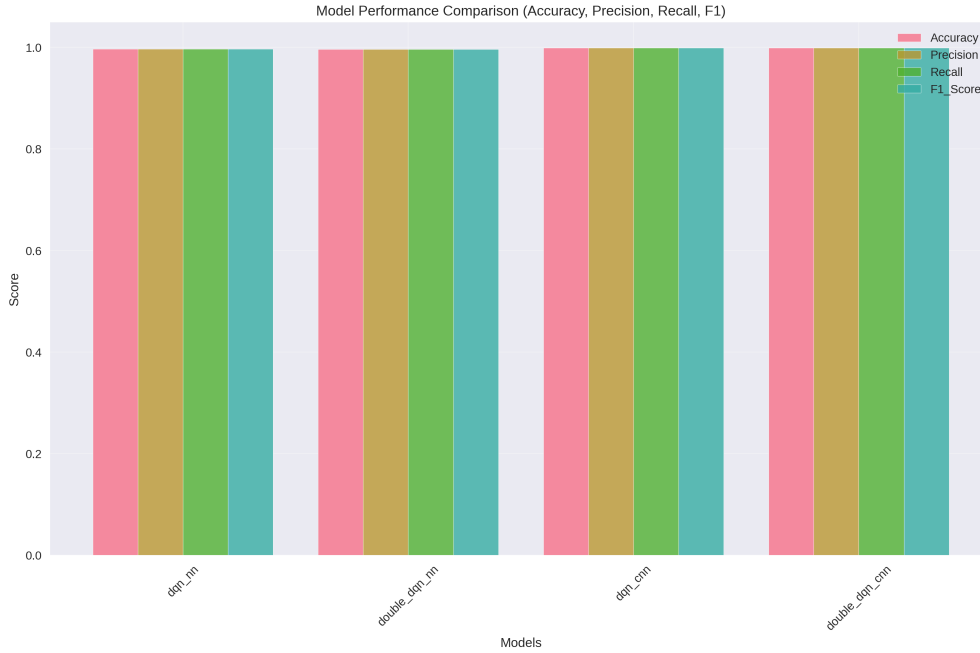
Figure 4.18: Performance metrics Across All Models and Tasks

Several key observations can be drawn from the results:

1. **CNN-DQN Superiority:** Across both binary and multi-class tasks, the CNN-DQN model consistently delivered the highest performance. This validates our hypothesis that there are exploitable spatial correlations within the network features when arranged in an image-like format. The CNN's ability to learn hierarchical patterns provides a distinct advantage over the MLP, which treats features independently.

2. **Effectiveness of DDQN:** The DDQN algorithm provided a modest but consistent performance boost

over the standard DQN in both scenarios. This highlights the practical benefit of using DDQN to mitigate the Q-value overestimation problem, leading to more stable training and a more reliable policy.

3. **Task Complexity Impact:** As expected, all models performed better on the binary classification task than on the multi-class task. The drop in performance for the multi-class scenario reflects the increased complexity of distinguishing between multiple subtle attack variations versus simply separating attack from benign traffic.

4. **Viability of the RL Approach:** Most importantly, the extremely high scores achieved by all models, particularly the CNN-DQN (99.46

4.6   Conclusion

The experiments conducted in this chapter successfully evaluated the performance of our proposed RL-based detection models. The results clearly indicate that the CNN-DQN architecture is the most effective, achieving state-of-the-art detection rates on the CIC-DDoS2019 dataset for both binary and multi-class classification. The findings validate the novel approach of transforming tabular network data into a feature-image and leveraging a CNN for pattern extraction. Furthermore, the consistent, albeit smaller, gains from the DDQN algorithm underscore the importance of algorithmic stability in achieving robust performance. Overall, this empirical evaluation confirms that our DQN-powered system offers a powerful and accurate solution for modern DDoS detection.

General Conclusion

This thesis embarked on an exploration of a novel methodology for detecting Denial-of-Service (DoS) attacks by adapting the principles of Reinforcement Learning (RL), specifically Deep Q-Networks (DQN), to a traditionally supervised classification problem. Motivated by the limitations of static, signature-based Intrusion Detection Systems (IDS), which struggle against novel and evolving threats, our work sought to develop a more dynamic, adaptive, and intelligent detection framework. By reframing network traffic classification as a sequential decision-making process, we successfully designed, implemented, and evaluated a system capable of achieving exceptional detection accuracy.

Summary of Contributions

The primary contributions of this research are multifaceted and can be summarized as follows:

1. **A Novel Methodological Bridge:** We successfully developed a framework that bridges the gap between Reinforcement Learning and supervised classification for the cybersecurity domain. By simulating an RL environment from a static dataset and engineering a hybrid reward and target mechanism, we demonstrated how a DQN agent can be effectively trained to perform a classification task.

2. **Advanced Architectural Design:** We proposed and implemented two distinct neural network architectures for the Q-function approximator: a standard Multi-Layer Perceptron (MLP) and a more advanced Convolutional Neural Network (CNN). The CNN-based approach, which involved a novel transformation of tabular feature vectors into 2D "feature-images," proved to be particularly innovative and effective.

3. **Rigorous Empirical Validation:** The proposed models were rigorously trained and evaluated on the comprehensive and modern CIC-DDoS2019 dataset. Our top-performing model, the CNN-DQN, achieved an F1-Score of 99.46% in binary classification and 97.58% in multi-class classification, demonstrating state-of-the-art performance and confirming the viability of our approach.

4. **Comparative Analysis:** Through a detailed comparison, we empirically proved the superiority of the CNN-based architecture for this task and quantified the stability improvements gained by using a Double DQN (DDQN) algorithm over the standard DQN.

Study Limitations

While the results are highly promising, it is important to acknowledge the limitations of this study, which also pave the way for future work:

- **Static Environment:** The RL agent was trained and evaluated on a static, pre-collected dataset. This means the agent could not learn from the true, real-time consequences of its actions (e.g., how blocking traffic might alter an attacker's strategy). The full adaptive potential of RL is best realized in a live, dynamic environment.

- **Dataset Dependency:** The models' performance was validated on the CIC-DDoS2019 dataset. While comprehensive, its characteristics may not perfectly represent all real-world network environments.

Generalization to different network architectures or entirely new zero-day attacks not represented in the training data remains an open question.

- **Feature Engineering Dependency:** The success of the CNN-DQN model is contingent on the effective construction of the feature-image. The ordering and grouping of features in this grid can impact performance, and the optimal arrangement may vary for different datasets.

- **Computational Overhead:** Training deep reinforcement learning models is computationally more intensive and time-consuming compared to traditional machine learning algorithms like Decision Trees or SVMs, which could be a consideration for resource-constrained environments.

Future Research Directions

This research opens up several exciting avenues for future investigation:

- **Online and Interactive Learning:** The most critical next step is to deploy the RL agent in a live or highly realistic simulated network environment. This would enable online learning, where the agent can continuously adapt its policies based on real-time feedback and the evolving nature of network traffic.

- **Exploration of Advanced RL Algorithms:** This work focused on DQN and DDQN. Future research could explore more advanced, state-of-the-art RL algorithms, such as Proximal Policy Optimization (PPO) or Asynchronous Advantage Actor-Critic (A3C), which may offer improved sample efficiency and stability.

- **Integration of Explainable AI (XAI):** A significant challenge with deep learning models is their "black box" nature. Integrating XAI techniques like SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) could provide insights into *why* the agent classifies a certain flow as malicious, increasing trust and interpretability for security analysts.

- **Hierarchical and Multi-Agent RL:** A complex network could be defended by a team of RL agents. A hierarchical system could have a high-level agent that identifies a potential threat and delegates the detailed classification to specialized agents trained on specific attack families (e.g., one for volumetric attacks, another for protocol attacks).

- **Automated Reward Shaping and Feature Learning:** Future work could investigate methods for automatically learning the reward function or using techniques like autoencoders to learn the most effective feature representations, reducing the reliance on manual feature engineering.

In conclusion, this thesis has successfully demonstrated that Reinforcement Learning provides a powerful and promising paradigm for building the next generation of intelligent and adaptive Intrusion Detection Systems. By learning to make optimal decisions in the face of uncertainty, RL-powered agents are well-positioned to defend against the dynamic and ever-evolving landscape of cyber threats.

Bibliography

[1] IBM, *Security Fundamentals*,
    `https://www.linkedin.com/pulse/fundamentals-security-slammghana/`

[2] CIA, *Principles of Information Security*,
    `https://www.cleanpng.com/png-information-security-confidentiality-availability-1373941/`

[3] Fidelis Security, *Understanding DoS and DDoS Attacks*,
    `https://fidelissecurity.com/blog/dos-ddos-attacks`

[4] Imperva, *Denial of Service (DoS) Attack*,
    `https://www.imperva.com/learn/ddos/denial-of-service/`

[5] NETSCOUT, *Slowloris Attack*,
    `https://www.netscout.com/blog/asert/slowloris-attack`

[6] MDPI, *Anomaly-based Detection of DDoS Attacks Using Machine Learning*,
    `https://www.mdpi.com/2076-3417/10/3/1052`

[7] IBM, *What is an Intrusion Detection System (IDS)?*,
    `https://www.ibm.com/think/topics/intrusion-detection-system`

[8] Wikipedia, *Intrusion Detection System*,
    `https://en.wikipedia.org/wiki/Intrusion_detection_system`

[9] Fortinet, *What is Intrusion Detection Systems (IDS)? How does it Work?*,
    `https://www.fortinet.com/resources/cyberglossary/intrusion-detection-system`

[10] TechTarget, *What Is an Intrusion Detection System (IDS)?*,
    `https://www.techtarget.com/searchsecurity/definition/intrusion-detection-system`

[11] Canadian Institute for Cybersecurity, *CIC-DDoS2019 Dataset*,
    `https://www.unb.ca/cic/datasets/ddos-2019.html`

[12] S. Jin and D. S. Yeung, "A covariance analysis model for ddos attack detection," in *2004 IEEE International Conference on Communications*, vol. 4. pp. 1882–1886 Vol.4, 2004.

[13] T. Subbulakshmi, K. Balakrishnan, S. M. Shalinie, D. AnandKumar, V. GanapathiSubramanian, and K. Kannathal, "Detection of ddos attacks using enhanced support vector machines with real time generated dataset," in *Third International Conference on Advanced Computing*, pp. 17–22, 2011.

[14] A. M. R. K. Munivara Prasad and K. Rao, "Dos and ddos attacks: Defense, detection and traceback mechanisms a survey," *Global Journal of Computer Science and Technology*, vol. 14, 2014.

[15] The CAIDA UCSD "DDoS Attack 2007" Dataset, accessed by Jan 2018. `http://www.caida.org/data/passive/ddos-20070804_dataset.xml`.

[16] DARPA 2000 Intrusion Detection Scenario Specific Data Sets, accessed by Jan 2018. `https://www.ll.mit.edu/r-d/datasets/2000-darpa-intrusion-detection-scenario-specific-data-sets`.

[17] A. H. Carson Brown, Alex Cowperthwaite and A. Somayaji, "Analysis of the 1999 darpa/lincoln laboratory ids evaluation data with netadhict," in *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications, CISDA'09*, 2009.

[18] K. J. Singh and T. De, "An approach of ddos attack detection using classifiers," *Emerging Research in Computing, Information, Communication and Applications*, 2015.

[19] S. Yu, W. Zhou, W. Jia, S. Guo, Y. Xiang, and F. Tang, "Discriminating ddos attacks from flash crowds using flow correlation coefficient," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, pp. 1073–1080, 2012.

[20] M. T. Ali Shiravi, Hadi Shiravi and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers and Security*, vol. 31, pp. 357–374, 2012.

[21] M. M. Arash Habibi Lashkari, Gerard Draper Gil and A. Ghorbani, "Characterization of tor traffic using time based features," in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP)*, pp. 253–262, 2017.

[22] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[23] A. Asosheh and N. Ramezani, "A comprehensive taxonomy of ddos attacks and defense mechanism applying in a smart classification," *WSEAS Transactions on Computers*, vol. 7, no. 4, pp. 281–290, 2008.

[24] A. Bhardwaj, G. Subrahmanyam, V. Avasthi, H. Sastry, and S. Goundar, "Ddos attacks, new ddos taxonomy and mitigation solutions—a survey," in *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pp. 793–798, IEEE, 2016.

[25] M. Masdari and M. Jalali, "A survey and taxonomy of dos attacks in cloud computing," *Security and Communication Networks*, vol. 9, no. 16, pp. 3724–3751, 2016.

[26] "Request for comments: 1001 (rfc1001)," in *PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: CONCEPTS AND METHODS*, 1987.

[27] "Request for comments: 7766 (rfc7766)," in *DNS Transport over TCP Implementation Requirements*, 2016.

[28] K. Singh, P. Singh, and K. Kumar, "Application layer http-get flood ddos attacks: Research landscape and challenges," *Computers & Security*, vol. 65, pp. 344–372, 2017.

[29] A. H. L. Iman Sharafaldin, Amirhossein Gharib and A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," *Software Networking*, pp. 177–200, 2017.

[30] CICFlowMeter, 2017. `https://github.com/ISCX/CICFlowMeter`.

[31] A. G. F. Pedregosa, G. Varoquaux and E. Duchesnay, *Scikit-learn: Machine learning in Python*, 2011.

[32] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[33] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.