



University of
Zurich^{UZH}

Novel Reinforcement Learning-powered Ransomware

Lukas Gübeli
Zurich, Switzerland
Student ID: 22-739-197

Supervisor: Dr. Alberto Huertas Celdran, Chao Feng,
Prof. Dr. Burkhard Stiller
Date of Submission: June 30, 2025

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich,

Signature of student

Lgibeli

Abstract

This bachelor's thesis explores the application of different reinforcement learning (RL) algorithms to optimise ransomware behaviour, with the goal of evading dynamic detection mechanisms, particularly in resource-constrained Internet of Things (IoT) environments. This work builds upon the RansomAI framework by addressing a notable gap in current research: the limited exploration of diverse RL strategies for malware optimisation. Instead of relying on a single approach, this work evaluates multiple approaches to identify the most effective strategies for maximising the stealth and operational impact of ransomware. The experimental setup includes a Raspberry Pi device acting as a spectrum sensor, infected with adaptive ransomware capable of changing its behaviour at runtime. Behavioural data collected from the device is used to train anomaly detection (AD) systems, while RL agents are trained to evade these models. The performance of the selected algorithms is evaluated using metrics such as evasion success rate, training convergence speed, and sample efficiency.

The results indicate that different RL techniques offer distinct advantages. Some methods excel in selecting optimal ransomware configurations within the first minute of training, while others demonstrate more stable convergence and consistent resource consumption. These insights contribute to the broader understanding of RL in adversarial machine learning and highlight the importance of aligning RL strategies with specific system constraints and performance goals.

Ultimately, this study demonstrates the significant potential of RL to enhance malware evasion and highlights the need for further research into resilient and adaptive defence mechanisms against intelligent cyber threats.

Zusammenfassung

Das Ziel dieser Bachelorarbeit ist es, die Anwendung verschiedener Reinforcement Learning (RL) Algorithmen zu untersuchen, um das Verhalten von Ransomware zu optimieren und ihre Fähigkeit zu verbessern, dynamische Erkennungsmechanismen zu umgehen. Insbesondere in ressourcenbeschränkten Internet of Things (IoT) Umgebungen. Aufbauend auf dem RansomAI-Framework schliesst diese Arbeit eine wichtige Forschungslücke, indem sie mehrere RL-Methoden untersucht, anstatt sich wie frühere Studien auf einen einzigen Ansatz zu konzentrieren.

Ziel ist es, die effektivsten Strategien zur Maximierung der operativen Auswirkungen von Ransomware zu identifizieren. Der Aufbau umfasst ein Raspberry Pi Gerät, das als Spektrumssensor fungiert und mit adaptiver Ransomware infiziert ist, die ihr Verhalten zum Zeitpunkt des Angriffs ändern kann. Die Verhaltensdaten des Geräts werden verwendet, um Anomalieerkennungssysteme zu trainieren, während RL-Agenten darauf trainiert werden, diesen Modellen zu entkommen. Die Leistung der Algorithmen wird anhand von Metriken wie der Erfolgsrate bei der Erkennungsumgehung, der Konvergenzgeschwindigkeit des Trainings oder auch der Stichprobeneffizienz bewertet.

Die Ergebnisse zeigen, dass verschiedene RL-Techniken deutliche Vorteile in Bezug auf Lerngeschwindigkeit, Ausweicherfolg und Recheneffizienz bieten. Einige Methoden zeichnen sich durch die Auswahl optimaler Ransomware-Konfigurationen innerhalb der ersten Minute des Trainings aus, während andere eine stabilere Konvergenz und einen gleichmässigeren Ressourcenverbrauch aufweisen. Diese Erkenntnisse tragen zu einem breiteren Verständnis von RL im Bereich des gegnerischen maschinellen Lernens bei und unterstreichen die Bedeutung der Anpassung von RL-Strategien an spezifische Systembeschränkungen und Leistungsziele.

Zusammenfassend zeigt diese Studie das erhebliche Potenzial von RL zur Verbesserung der Malware-Abwehr und unterstreicht die Notwendigkeit weiterer Forschung im Bereich robuster und adaptiver Abwehrmechanismen gegen intelligente Cyber-Bedrohungen.

Acknowledgments

I would like to express my sincere gratitude to my thesis supervisors, Dr. Alberto Huertas Celdrán and Chao Feng. Their valuable suggestions, insightful feedback, and continuous support throughout the entire thesis process were instrumental in shaping the quality of this work. I especially appreciated the clear guidance and the excellent and timely communication during the final stages of the project.

Furthermore, I would like to thank Prof. Dr. Burkhard Stiller for giving me the opportunity to conduct my bachelor's thesis at the Communication Systems Group (CSG).

A special thanks goes to Mercedes Locher for her kind support and encouragement throughout the writing process.

Contents

Declaration of Independence	i
Abstract	ii
Zusammenfassung	iii
Acknowledgments	iv
1 Introduction	1
1.1 Motivation	2
1.2 Description of Work	2
1.3 Thesis Outline	3
2 Background	5
2.1 Electrosense	5
2.2 Behavioural Fingerprinting	6
2.3 Ransomware	6
2.3.1 Types of Ransomware	7
2.4 Reinforcement Learning	8
2.4.1 Exploration vs Exploitation	9
2.4.2 Markov Decision Processes	9
2.4.3 Dynamic Programming	10
2.4.4 Monte Carlo Methods	10
2.4.5 Temporal Difference	11
2.4.6 Different Types of Reinforcement Learning	12
2.4.7 From Reinforcement Learning to Neuroscience	15
3 Related Work	17
3.1 AI-Powered Malware	17
3.1.1 Surveys	18
3.2 Reinforcement Learning in Cyber Security	20
3.3 Summary	22
4 Scenario	25
4.1 Evaluation of Reinforcement Learning Algorithms	25
4.1.1 Comparison of Basic Reinforcement Learning Algorithms	26
4.1.2 Comparison of Deep Reinforcement Learning Algorithms	29
4.1.3 Selection of the Reinforcement Learning Algorithms	33
4.1.4 Refinements and Extensions	34

4.2 Environment Setup	34
4.2.1 Ransomware	35
4.2.2 Fingerprint Data	36
5 Implementation	37
5.1 Reinforcement Learning Environment	37
5.1.1 Anomaly Detection	38
5.1.2 Reward Function	38
5.1.3 Agent Functionalities	39
5.1.4 Processed Input Data	40
5.2 Prototypes	42
5.2.1 DDQL Prototype	42
5.2.2 SARSA Prototype	45
5.2.3 PPO Prototype	48
6 Comparison	55
6.1 DDQL Performance differences	56
6.2 Comparison of DQL and DDQL	58
6.3 SARSA with- and without Neural Network	59
6.4 Comparison of DDQL and PPO	60
6.4.1 Accuracy and training speed	60
6.4.2 Consumption Differences	61
6.5 Comparisons of additional training Hardware	63
6.5.1 Performance Differences	65
6.5.2 Consumption Differences	65
7 Final Considerations	67
7.1 Limitations	67
7.2 Summary and Conclusions	68
7.3 Future Work	69
Bibliography	71
Abbreviations	77
List of Figures	79
List of Tables	81
List of Equations	83
List of Algorithms	85
List of Listings	87
A Contents of the Repository	89

Chapter 1

Introduction

During the last 25 years, the usage of IoT devices has grown at an extremely rapid pace. It is estimated that by 2033, there will be approximately 40 billion IoT devices, which is more than double the 16 billion devices reported in 2023 [1]. However, the growth of IoT devices also increases the potential attack surface for cyber threats, such as the leakage of sensitive data, unauthorised network access, or denial of service (DoS) attacks. IoT devices are particularly vulnerable to the aforementioned threats, as they often lack the necessary security measures. Many devices violate the privacy of users by mishandling personal information, have low password protection, and do not implement encrypted communication protocols [2]. This significant lack of security arises primarily because manufacturers prioritise keeping costs low to remain competitive in the market. As a result, security features are often neglected or underdeveloped [3]. Although some countries, such as the United Kingdom, have introduced regulations mandating basic security standards for IoT systems [4], most IoT devices still pose a significant threat. This highlights the urgent need for robust cyber security measures in the IoT domain.

To establish a robust cyber security system, an increasing number of researchers and organisations are integrating artificial intelligence (AI) into their defences. Keeping AI-based defence systems up-to-date is particularly important, as AI can also be used for offensive cyber attacks. A fundamental requirement for training an effective defence system is a thorough understanding of the attacks themselves [5].

Previous work conducted by the CSG at the University of Zurich focused on this aspect [6-8], implementing RL-based ransomware to target an IoT device equipped with spectrum sensors. The results showed that AI can be used to optimise ransomware attacks, highlighting the urgent need for improved countermeasures. However, the study was subject to several limitations that affected the adaptability and effectiveness of the proposed approach. A key limitation was its reliance on a single RL algorithm, without exploring alternative methods that could offer better generalisation, robustness, or sample efficiency.

To address this gap, this thesis explores and compares different RL algorithms to assess their performance and suitability for optimising ransomware attacks on IoT devices. By analysing different RL methods, the thesis seeks to provide deeper insight into their respective advantages and disadvantages from an offensive perspective. The aim is to

identify strategies that enhance the effectiveness, adaptability, and stealth of AI-driven ransomware, thus contributing to a more comprehensive understanding of the potential of AI in modern cyber threats.

Although this may seem alarming and ethically controversial, previous studies (e.g. [6–8]) and this thesis strive to deepen our understanding of current and emerging threats. By investigating the offensive potential of AI-driven cyber attacks, this research contributes to strengthening cyber security defences and developing more resilient systems, ultimately helping to prevent future AI-driven attacks.

1.1 Motivation

Recent studies indicate that AI-driven malware can evade traditional security mechanisms using methods such as adversarial training and dynamic evasion strategies [5]. Surveys of AI-based cyber threats categorise these advances into several areas, including next-generation malware, password-based attacks, and adversarial manipulation of AI models [9]. Within this evolving landscape, ransomware has emerged as a particularly destructive cyber threat, with a 232% increase in attacks between 2019 and 2021, and global cybercrime costs projected to reach 10.5 trillion by 2025 [10]. These trends highlight the critical need to understand how AI techniques, particularly RL, contribute to the evolution of ransomware strategies.

While existing research has predominantly concentrated on static evasion techniques, the exploration of malware capable of dynamically adapting during execution remains limited. Furthermore, most studies employ value-based RL methods, such as Deep Q-Learning, while alternative approaches, including policy-based and actor-critic models, remain largely unexplored.

To overcome this limitation, this thesis investigates novel RL-based ransomware methods capable of adapting in real-time to evade security defences. Through experimentation with different RL strategies, this work aims to improve the effectiveness of AI-driven ransomware and provide a comparative analysis of their performance.

1.2 Description of Work

This bachelor’s thesis builds on the work presented in the previous master’s thesis by M.Luechiger [8], and aims to develop more advanced and effective versions of RL algorithms to optimise ransomware attacks on IoT devices. The goal is to train a ransomware agent capable of performing malicious actions while avoiding detection. The agent will operate in a simulated environment, learning to maximise a reward function by exploiting system vulnerabilities and minimising the risk of being detected.

The results of this research will be evaluated and compared with those of the referenced master’s thesis [8], using a range of performance metrics to assess the effectiveness and impact of different RL algorithms on resource-constrained spectrum sensors.

The study begins with an analysis of various state-of-the-art RL algorithms. Based on this analysis, a selection of algorithms will be chosen for implementation. Each selected algorithm will be implemented and fine-tuned prior to a detailed performance evaluation. The comparison will be based on several key indicators, including algorithm accuracy, training speed, sample efficiency, resource consumption, convergence speed, and overall reward performance, while taking into account the specific characteristics and implementation details of each algorithm.

The primary objective of this thesis is to determine whether an alternative RL algorithm can outperform the Deep Q-Learning (DQL) approach used in the previous study and to examine the strengths and limitations of applying RL to the broader domain of ransomware and malware.

1.3 Thesis Outline

The thesis begins in Chapter 2 with a concise explanation of the fundamentals of Electrosense and behavioural fingerprinting. It then shifts focus to the topic of malware, providing an overview of its various types and their growing relevance in today's digital environment. The chapter concludes with a comprehensive introduction to RL, covering both its foundational concepts and more advanced topics, including the key differences between various RL algorithms.

To explore recent advancements and ongoing challenges in the field of AI-powered malware, Chapter 3 conducts a survey of the existing literature. Special attention is given to research that focuses on the application of RL to malware, as well as an examination of key RL algorithms.

Chapter 4 outlines the specific experimental scenario addressed in this thesis. It details the tasks performed and describes the structure of the simulated environments used for training and evaluation.

The implementation itself is covered in Chapter 5. The selected algorithm is designed to train ransomware to achieve optimal performance while avoiding detection. The chapter also defines and explains the features of the spectrum sensors affected by the ransomware and looks at the performance results of the different approaches.

In Chapter 6 the performance of the RL algorithms is compared with one another as well as with the results of the previous study [8]. This evaluation highlights whether the set objectives were achieved and provides explanations for any observed discrepancies.

Finally, Chapter 7 concludes the thesis by summarising the key findings and contributions. In addition, it discusses areas for future research, identifying open questions and suggesting potential directions for further exploration in the field.

Chapter 2

Background

The background provides an explanation of the basic concepts underlying this thesis. Its purpose is to establish a comprehensive understanding of the theoretical framework, which is essential for grasping the scenarios, as well as for interpreting the findings presented in the subsequent chapters. This chapter is divided into four main sections. It begins with an overview of Electrosense, which lays the foundation for the concepts explored throughout this thesis. The second section discusses behavioural fingerprinting, outlining its principles and applications within the scope of this research. This is followed by a comprehensive introduction to ransomware, covering its different types and highlighting its relevance in modern society. The chapter concludes with an in-depth exploration of RL, introducing its fundamental concepts along with a brief overview of some of the most widely used RL algorithms.

2.1 Electrosense

Electrosense is a crowdsourced platform designed for efficient and scalable radio frequency spectrum monitoring in different regions of the world. It uses low-cost hardware, such as Raspberry Pi-based sensors equipped with software-defined radio (SDR) front-ends, to collect real-time spectrum data. This data is subsequently processed and made available via an open API, enabling researchers and developers to analyse spectrum usage patterns, detect anomalies, and develop new applications.

The Electrosense architecture is comprised of three principal components: a distributed network of sensors that collect spectrum data, a centralised controller that manages sensor configurations and measurement campaigns, and a backend system responsible for data aggregation, processing, and distribution. The backend incorporates multiple layers of processing, including both real-time and batch analysis, to extract meaningful insights from the collected data.

To ensure security and reliability, Electrosense employs encrypted data transmission through TLS and implements safeguards against unauthorised access [11]. Despite these measures, spectrum sensors remain vulnerable to various cyber security threats, such as

identity spoofing, malware infection, and data tampering attacks. In response, researchers have explored the use of behavioural fingerprinting and machine learning techniques to detect and mitigate these risks, demonstrating the potential of Electrosense not only as a spectrum monitoring tool but also as a valuable testbed for security research [12].

2.2 Behavioural Fingerprinting

Behavioural fingerprinting is a dynamic cyber security approach that focusses on monitoring and analysing device behaviour to detect malware and identify anomalies. Unlike traditional static methods that rely on known malware signatures, behavioural fingerprinting is effective against invisible, encrypted, or obfuscated malware. It operates by establishing a baseline of normal device behaviour and employing machine learning (ML) or deep learning (DL) models to identify deviations caused by infections or irregularities.

The process involves collecting data during normal device operation to create a baseline dataset. If the goal is to classify specific types of malware, additional data from infected devices is required. Behavioural fingerprinting is particularly useful for IoT devices, which often have predictable patterns of behaviour. Its key strength lies in its ability to detect malware regardless of code obfuscation, as the underlying behaviour remains consistent.

This technique is applied in two main scenarios: device identification (determining device type, model, or individual identity) and misbehaviour detection (identifying anomalies caused by cyber attacks or hardware failures). Data can be collected externally, such as network traffic, or internally, including hardware performance counters, resource usage and system calls [13].

2.3 Ransomware

Since the first wave of modern ransomware in 2005, the popularity of this attacking method has reached alarmingly high levels [14]. In 2023 every third cyber incident was attributed to a ransomware attack [15].

The ransomware lifecycle begins when the malicious code enters the victim's system through methods such as email attachments, drive-by downloads, or code droppers. Once inside, it performs several actions, which include generating a unique computer ID, disabling shadow copies, and ensuring persistence at startup. Afterwards, it contacts a Command and Control (C&C) server to obtain an encryption key. The ransomware then searches for specific file types (e.g., PDFs, documents, images) and encrypts them, renaming, and deleting the originals. Finally, a ransom demand is displayed to the victim, either as a text file or on the desktop screen.

2.3.1 Types of Ransomware

There are two main types of ransomware: cryptographic ransomware and locker ransomware. Cryptographic ransomware encrypts files and data on the victim's system, rendering them inaccessible. To regain access, victims have to pay a ransom for the decryption key [14]. This type of ransomware employs different encryption methods, including symmetric, asymmetric, or hybrid encryption to lock victims' files. Symmetric encryption is faster and more resource-efficient, but requires the attacker to secure the key, often using Advanced Encryption Standard (AES). Asymmetric encryption, typically using the Rivest-Shamir-Adleman (RSA) algorithm, employs a public/private key pair, preventing key exposure, but it is slower when dealing with large files. Hybrid encryption combines both methods, first by encrypting files with a symmetric key and then by securing the key with asymmetric encryption, providing efficiency and security without the need for a C&C server connection [16].

The second type, locker ransomware, locks users out of their system or device entirely, typically displaying a ransom demand on the screen. Unlike cryptographic ransomware, it does not encrypt the data on the system, making it quicker and easier to execute. This poses a particular threat to IoT devices, wearables, and mobile devices, where the limited manoeuvrability of these devices increases vulnerability.

Another form of malware that acts as ransomware is scareware, which relies on deceptive tactics to manipulate users. It creates the false impression that a system is compromised or that the victim is involved in illegal activity. Thus, inducing fear to pressure users to pay a fee to solve the perceived problem. Common scareware variants include fraudulent antivirus programs that simulate virus scans, falsely claiming to detect malware, and encouraging users to purchase a fake removal tool. In some cases, scareware serves as a distraction from more serious cyber threats, such as ransomware. Cybercriminals use social engineering techniques to make the scareware appear convincing, contributing to its effectiveness in deceiving victims [14].

Lastly, a notable type of ransomware that has grown in popularity is Ransomware-as-a-Service (RaaS). RaaS operates as a business model in which cybercriminals distribute ransomware to affiliates who execute attacks in exchange for a share of the profits. This approach reduces the technical barrier for attackers, allowing individuals with minimal hacking skills to deploy ransomware. RaaS operators provide ready-to-use ransomware kits, payment processing, and even customer support to maximise their revenue. The proliferation of this model has led to an increase in ransomware incidents, as it enables widespread distribution and frequent updates to evade detection. By outsourcing the execution of attacks, cybercriminals increase efficiency while reducing their own risk of exposure [17].

2.4 Reinforcement Learning

The concept of RL originated in the psychology of animal learning, specifically in the context of trial-and-error behaviour. In the late 1980s, this idea was formalised and began to take shape as a computational approach. RL is now recognised as the third paradigm of ML, along with supervised and unsupervised learning.

The primary goal of RL is to maximise a numerical reward signal. In this framework, an agent interacts with its environment and learns through trial and error, receiving delayed rewards for its actions. A high reward signals that the action was beneficial, while a low reward indicates the opposite. This means that the RL agent must be able to learn independently of its own experiences and adapt its behaviour to achieve better results over time.

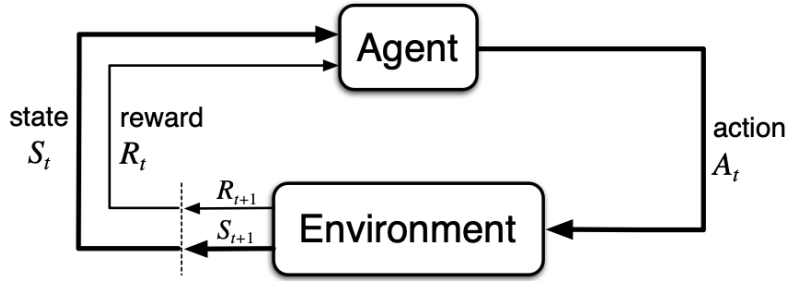


Figure 2.1: Overview of the RL structure [18]

All RL agents share the fundamental characteristic of being goal-oriented, seeking to achieve predefined objectives through interaction with their environment. To accomplish this, they must have an understanding of their environment and the ability to select actions that influence it in ways that align with their objectives [18].

Every RL system consists of four fundamental components [19]:

- **The Policy** defines the agent’s strategy for choosing actions based on its current state. It serves as a decision-making function and can be deterministic, usually indicated by μ :

$$A_t = \mu(S_t), \quad (2.1)$$

or it can be stochastic and will then be indicated by π :

$$A_t \sim \pi(\cdot | S_t). \quad (2.2)$$

A well-optimised policy ensures the agent selects actions that maximise long-term rewards [18]. As the policy functions like the agent’s brain, some literature uses the words ”policy” and ”agent” interchangeably.

- **The Value Function** estimates the expected benefit for the agent of being in a particular state. It provides a long-term perspective by predicting the cumulative rewards the agent can expect from a given state, rather than focussing solely on immediate outcomes.
- **The Reward Signal** provides feedback to the agent regarding its performance in relation to the established goal. It is a value issued after each action, which guides the agent's learning process by reinforcing behaviours that lead to higher rewards while discouraging less effective actions.
- **The Environment** encompasses all elements with which the agent interacts. It defines the states, actions, and rules that govern how the agent's actions affect the world. Serving as a source of feedback the environment provides the state information and reward signals based in response to the agent's actions.

The four elements considered constitute the foundation of RL. This allows agents to learn, adapt and improve their performance over time through interaction and feedback from their environment [18].

2.4.1 Exploration vs Exploitation

A fundamental trade-off in RL is between exploration and exploitation. Exploration involves trying out new actions that are not yet fully understood, aiming to discover potential rewards and improve the agent's understanding of the environment. In contrast, exploitation focusses on leveraging the knowledge already gained to maximise immediate rewards by selecting actions that are already known to be optimal. Finding the right balance between these two is crucial: excessive exploration may lead to wasted time and resources on suboptimal actions, while excessive exploitation can cause the agent to miss better long-term strategies or rewards [20, 21]. A common approach to balancing exploration and exploitation is the ϵ -greedy method. In this method, a static ϵ is used between $[0, 1]$ to defines its probability of choosing a non-greedy action (exploration by choosing a random action) [18, 21].

2.4.2 Markov Decision Processes

The Markov Decision Process (MDP) is a mathematical framework that allows decision-making under uncertainty. In RL, the trade-off between exploration and exploitation introduces uncertainty about the agent's next step. MDPs provide a structured way to represent this uncertainty and enable agents to make informed decisions by considering both the immediate and long-term consequences of their actions.

In finite MDPs, the system consists of an agent, an environment, and a reward structure that the agent aims to maximise, with the sets of states, actions, and rewards being finite. The Markov property states that the probabilities of transitioning to a given

state and receiving a reward at any given time depend solely on the preceding state and action, with no influence from earlier states or actions. Since the agent's objective is to maximise cumulative reward over time rather than immediate returns, future rewards must be adjusted to account for their present value. This adjustment is achieved using a parameter $\gamma \in [0, 1]$, known as the discount rate, which determines the importance of future rewards relative to immediate ones. At time t , the agent selects an action A to maximise the expected discounted return, which is defined as:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \quad (2.3)$$

where R_{t+k+1} represents the reward received $k + 1$ time steps into the future, and γ determines the weight of future rewards relative to immediate rewards [18].

2.4.3 Dynamic Programming

Dynamic Programming (DP) is a powerful framework to solve complex optimisation problems by decomposing them into smaller, overlapping subproblems. Within the context of RL, DP serves as a fundamental tool for solving finite MDPs. DP employs value functions to systematically evaluate and improve policies, ultimately leading to the discovery of the optimal policy for a given MDP.

The two core components of DP in RL are policy evaluation and policy improvement. Policy evaluation iteratively computes the value function for a given policy, quantifying the expected long-term reward from each state. Subsequently, policy improvement uses this value function to derive a better policy by selecting actions that maximise the expected reward. By alternating between these steps, DP methods, such as policy iteration and value iteration, can efficiently compute optimal policies and value functions [18].

2.4.4 Monte Carlo Methods

Monte Carlo (MC) methods offer a valuable approach to solving RL problems by relying on sample-based learning to approximate value functions and refine policies. Unlike DP, which requires explicit knowledge of the environment, MC methods learn directly from experience, either from actual interaction with the environment or from simulated samples. This ability to operate without a model makes MC methods particularly attractive for scenarios where explicit modelling of the environment is not feasible.

MC methods estimate value functions by averaging the returns obtained from sampled episodes. Returns are computed for each state-action pair by observing complete episodes and recording the cumulative rewards received. These methods are primarily suited for episodic tasks, where each interaction with the environment naturally concludes after a finite sequence of steps. Consequently, updates to value estimates and policies occur only

at the end of each episode, rendering the learning process incremental in an episode-by-episode sense, but not in an online or step-by-step manner.

A central advantage of MC methods is their ability to learn directly from raw experience without relying on a predefined model of the environment. This makes them suitable for environments where it is easy to generate sample transitions but difficult to obtain explicit transition probabilities. In addition, according to the law of large numbers, sample averages of returns converge to their true values as more episodes are observed. A basic MC method for a non-stationary environment may be structured as follows:

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)], \quad (2.4)$$

where G_t is the actual return following time t , and α is a constant step size parameter. In general, MC methods provide a powerful framework for RL, allowing agents to improve their decision making through interaction and experience. Their reliance on sampling, rather than precise environmental dynamics, makes them a versatile approach to solving complex tasks in uncertain or partially understood environments [18].

2.4.5 Temporal Difference

Temporal Difference (TD) learning is a fundamental concept in RL that merges key aspects of MC methods and DP [18]:

- **MC Method Features:** TD methods are capable of learning directly from raw experience, without requiring a complete or accurate model of the environment.
- **DP Features:** TD methods update their value estimates by partially relying on previously learnt estimates, rather than waiting for the full return at the end of an episode.

This hybrid functionality enables TD methods to efficiently address the prediction problem by estimating the value function v_π for a given policy π . Unlike MC methods, which require waiting until the end of an episode to compute the return G_t before updating the value estimates, TD methods update their predictions incrementally at each time step. One of the simplest forms of TD learning, known as TD(0) or one-step TD, is defined as follows:

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)], \quad (2.5)$$

In comparison to the MC equation in 2.4.4, the TD method updates its estimates immediately following the transition to S_{t+1} and receipt of R_{t+1} . This immediate feedback mechanism, which eliminates the need to wait until the end of an episode, makes TD methods significantly more efficient.

2.4.6 Different Types of Reinforcement Learning

The primary goal of RL, as with all other learning systems, is to solve a specific problem. The suitability of specific RL algorithms varies depending on the nature of the problem at hand. Broadly, RL can be categorised into four main approaches: Model-Based RL (Chapter 2.4.6.1), Value-Based RL (Chapter 2.4.6.2), Policy-Based RL (Chapter 2.4.6.3) and Hybrid Approaches, also known as Actor-Critic Methods (Chapter 2.4.6.4). Each of these approaches has its own set of advantages and disadvantages, making them better suited for different types of problems and environments [22, 23].

2.4.6.1 Model-Based Reinforcement Learning

Model-Based RL focusses on creating an internal model of the environment, allowing the agent to simulate the results and plan its actions before interacting with the real environment. This approach enables agents to make decisions more efficiently by predicting the consequences of their actions. It is particularly useful when interacting directly with the environment is expensive or risky.

Advantages:

- Higher sample efficiency, since the agent can simulate actions
- Enables more effective planning in environments with structured transitions and therefore more suitable for environments where real-world experimentation is costly or impractical.

Disadvantages:

- Requires an accurate model of the environment as the performance heavily depends on it.
- Building an accurate model of the environment can be complex and computationally expensive.

2.4.6.2 Value-Based Reinforcement Learning

Value-Based RL revolves around estimating the value of individual states or state-action pairs to guide the agent's decisions. By evaluating these values, the agent is able to select the actions that are expected to maximise long-term rewards. Techniques such as Q-learning are common examples of this approach [22].

Advantages:

- Straightforward implementation, especially for discrete state and action spaces.
- Efficient in environments where state transitions and rewards are relatively stable.

Disadvantages:

- Struggles with high-dimensional or continuous action spaces.
- Often, significant exploration is required to achieve reliable value estimates, which can be computationally intensive.

Q-Learning

Q-Learning is an off-policy, model-free RL algorithm based on TD learning. It derives its name from its main goal: learning the quality (Q-value) of actions in particular states. This value is formally defined by [18]:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{\alpha} Q(S_{t+1}, \alpha) - Q(S_t, A_t) \right]. \quad (2.6)$$

"The variables used in the above equation are defined as follows: s represents the current state, while a denotes the current action taken. Upon executing an action a in state s , the agent receives a reward r . The next state after taking action a is denoted as s' , and the action chosen in this new state is a' . The learning rate, symbolised as α , determines the extent to which newly acquired information should overwrite the existing knowledge. Lastly, γ represents the discount factor, which defines the significance attributed to future rewards."¹

Q-Learning offers several advantages, such as its simplicity and effectiveness in environments with small action spaces. In addition, it does not require a model of the environment, which makes it easy to implement in diverse scenarios. Nevertheless, Q-Learning encounters significant challenges when dealing with large or continuous state spaces, as it relies on Q-tables to store action-value pairs. This approach can become inefficient in terms of memory usage, especially in large environments [22].

To overcome these limitations, DQL integrates a neural network to approximate the Q-values, eliminating the need for explicit Q-tables. This shift enables DQL to efficiently handle large or continuous state spaces, making it a more scalable and effective solution for complex problems. As a result, DQL greatly enhances the learning process in RL applications, leading to more robust and adaptable models that can generalise in diverse environments [24].

¹This text was generated using generative artificial intelligence

2.4.6.3 Policy-Based Reinforcement Learning

Policy-Based RL directly optimises the policy by adjusting it to maximise the expected cumulative reward. Instead of estimating value functions, the agent learns the probability distribution over actions for each state. This approach is commonly used in continuous action spaces and complex environments [22, 23].

Advantages:

- Offers greater effectiveness in handling continuous or high-dimensional action spaces.
- Optimises stochastic policies directly, enabling better exploration and adaptability.

Disadvantages:

- May converge to suboptimal policies if not properly tuned.
- Generally, requires larger amounts of data and computational resources.

REINFORCE Algorithm

The REINFORCE algorithm is a policy-based RL method that optimises an agent's policy applying policy gradient techniques. Unlike value-based approaches such as Q-learning, REINFORCE directly parametrises and updates a stochastic policy $\pi_\theta(A_t|S_t)$ with the objective to maximise the expected cumulative reward. It follows a Monte Carlo strategy, i.e. updates only occur after an entire episode has been completed. The algorithm samples a trajectory, calculates the total return $G = \sum_{k=t+1}^T \gamma^{k-t-1} R_k$, where γ is the discount factor, and then adjusts the policy parameters θ using gradient ascent. The core update rule for REINFORCE is [18]:

$$\theta_{t+1} \doteq \theta_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)} \quad (2.7)$$

Here α is the learning rate and the fraction represents the policy gradient, which scales the parameter update by the logarithmic probability gradient of the selected action A_t given the state S_t . This equation ensures that actions leading to higher rewards are more likely to be reinforced, while those resulting in lower returns gradually diminish in probability. However, since REINFORCE relies on full episodes for updates, it is prone to high variance. Techniques such as baseline subtraction can help reduce this issue and improve stability. Despite its limitations, REINFORCE remains a fundamental approach in policy gradient methods.

2.4.6.4 Actor-Critic Reinforcement Learning

Hybrid Approaches, such as Actor-Critic Methods, combine the strengths of value-based and policy-based approaches. The actor is responsible for selecting actions according to a parameterised policy, while the critic evaluates the value of those actions to guide policy improvement. This dual architecture creates a balance between policy optimisation and value estimation [22].

Advantages:

- Combines the benefits of policy-based and value-based methods, providing greater stability and efficiency.
- Effective in handling complex environments with continuous or high-dimensional action spaces.

Disadvantages:

- More complex implementation and training due to the interaction between the actor and critic components.
- Requires careful tuning of hyper parameters to avoid instability or divergence during training.

A prominent example of an actor-critic algorithm is Soft Actor-Critic (SAC). It is an advanced RL algorithm that combines elements of policy optimisation and value-based methods. Its uniqueness lies in the idea of maximum entropy RL, which encourages exploration by promoting stochastic policies. Unlike traditional methods that focus solely on reward optimisation, SAC balances reward maximisation with entropy maximisation of the policy. This dual objective leads to more stable learning and better performance in continuous action spaces. Moreover, SAC demonstrates high sample efficiency and performs well in high-dimensional environments, making it a widely adopted approach in contemporary RL tasks [22].

2.4.7 From Reinforcement Learning to Neuroscience

In experimental research conducted by Schultz and colleagues [25], monkeys were trained to associate an instructional signal with a delayed reward, such as a drop of juice. In early stages of learning, dopamine neurons fired strongly upon receiving the reward, signalling an unexpected reward and a high TD error due to inaccurate predictions. As learning progressed, the dopamine response shifted to the instruction signal, indicating that the reward prediction had been updated backwards in time. When the reward was fully predicted, dopamine activity at the time of the reward decreased to baseline, reflecting a TD error of zero. However, when the expected reward was omitted, dopamine activity at the usual reward time dropped below the baseline, corresponding to a negative TD

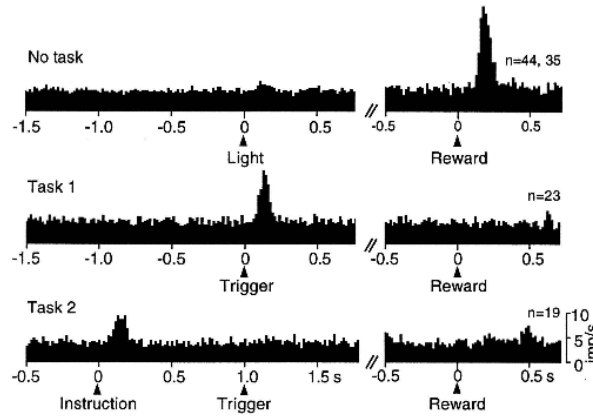


Figure 2.2: Response of dopamine neurons [18]

error. These observations demonstrate that dopamine neurons encode errors in reward prediction, which is consistent with the principles of TD learning in RL.

The correspondence between TD error and dopamine activity illustrates how TD error in RL reflects the phasic responses of dopamine neurons during reward-based tasks. Initially, dopamine neurons show strong activity in response to unexpected rewards, corresponding to a high TD error due to inaccurate predictions. As the value function is acquired and predictions improve, dopamine activity shifts to cues that reliably predict rewards, reflecting the behaviour of the TD error in accurately modelling future outcomes. Once a reward is fully predicted, both dopamine activity and TD errors decrease to zero. Conversely, when an expected reward is omitted, both dopamine responses and TD error exhibit negative responses, signalling a prediction mismatch. These findings highlight a close relationship between dopamine-driven learning in the brain and TD learning algorithms, reinforcing the idea that the brain may employ TD-like processes to support adaptive learning and decision-making [18].

Chapter 3

Related Work

This chapter is structured into two main sections. The first section provides an overview of the current state of knowledge on AI-powered malware, with a particular focus on ransomware. To provide a broader perspective, other forms of smart malware are also examined. This analysis is based on various surveys which are described in greater detail in Chapter [3.1.1](#). The literature review was conducted using the snowball method (both backward and forward), taking the prior study underlying this thesis as a starting point [\[8\]](#). Additionally it was supplemented by targeted keyword searches using terms such as "AI in malware", "offensive AI" and "RL malware".

The second section focusses on RL algorithms, with an emphasis on their key differences. Recent developments in RL are explored, along with a discussion of their strengths and limitations. Particular attention is given to offensive AI techniques that utilise RL, offering a detailed analysis of current practices and techniques in this domain.

The chapter ends by summarising the main insights obtained from the literature review, ensuring coherence throughout the chapter and reinforcing the relevance of the discussed material. The main objective is to provide a concise yet comprehensive overview of AI-enabled malware and offensive AI strategies, offering both clarity and depth for readers engaging with this emerging field.

3.1 AI-Powered Malware

When discussing the role of AI in the context of malware, the focus is typically placed on system hardening, where environments are strengthened by integrating AI in various domains. These improvements usually benefit security teams through the "3R model", which emphasises on increased robustness, faster response times, and improved resilience [\[26\]](#). However, recent advancements have also drawn attention to the offensive applications of AI within cyber security, highlighting its potential to develop more sophisticated and adaptive threats. It goes without saying that these discussions are framed within a responsible research context, with careful consideration given to ethical implications and the justification for studying adversarial capabilities.

3.1.1 Surveys

This section provides an overview of four key surveys that examine the role of AI in the context of malware, with a particular focus on offensive applications and detection mechanisms. Each survey offers unique insights into the current landscape of AI-powered malware and its implications for cyber security, highlighting various dimensions of this rapidly evolving field.

Building on the foundational work of one of the theses underlying this study [7], and given the rapid advances in AI-driven cyber security, this section does not dive into the historical background but instead concentrates on the latest findings and trends. All analysed sources are relatively recent, reflecting the ongoing evolution of the field. The earliest paper included, published in 2020, focusses on AI-based cyber threats [9]. In contrast, a more recent 2024 study extends the discussion to sector-specific impacts of AI-powered cyber threats [27]. To further explore the latest developments, a 2024 paper titled *Artificial Intelligence in Cyber security: A Review and a Case Study* [28] also examines some cyber threats and provides additional information. Lastly, to gain a better understanding of the current principles and mechanisms of ransomware, this section consults another 2024 study titled *AI-Based Ransomware Detection: A Comprehensive Review* [10], which approaches the topic from a defensive perspective.

The 2020 survey outlined five primary types of AI-powered cyber threats, along with specific examples for each category [9]:

- **Next-Generation Malware:** AI-enhanced malware designed to increase its effectiveness and adaptability.
- **Voice Synthesis:** Utilises AI to generate realistic voice recordings for fraudulent activities, such as impersonation.
- **Password-Based Attacks:** Employs ML to crack passwords more efficiently than traditional techniques.
- **Social Bots:** AI-powered bots used to influence the narratives of social networks and spread disinformation.
- **Adversarial Training:** Methods in which attackers corrupt AI models by injecting manipulated inputs to disrupt their accuracy.

Although some specific examples may no longer be fully applicable, these categories remain highly relevant. Moreover, the study introduced an AI-based cyber threat framework that deconstructs the cyber attack process into sequential stages to improve detection, response, and mitigation strategies. This framework consists of seven stages, divided into three main phases. Phase one includes reconnaissance and weaponisation. Phase two encompasses delivery, exploitation, and installation. The final phase covers C&C and targeting. The integration of AI at each stage increases the adaptability and complexity of attacks compared to traditional cyber threats. By applying this structured framework,

security professionals can better align their defences with the evolving AI-driven attack landscape, ensuring a more proactive and strategic approach to threat mitigation [9].

The survey on AI-based cyber threats not only explores some of the most common AI-driven cyber risks, but also examines their impact on specific industries [27].

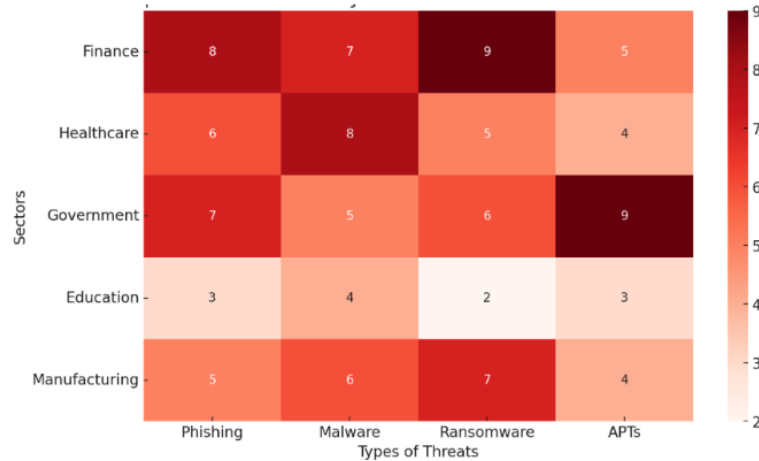


Figure 3.1: Heat map for the impact of AI-powered cyber threats across different sectors [27]

As shown in Figure 3.1 and detailed in the study, sectors such as finance, healthcare, and government are particularly vulnerable to these threats. The four main cyber threats discussed are phishing, malware, ransomware, and advanced persistent threats. Industries such as finance are particularly vulnerable, as successful cyber attacks can have significant economic and operational consequences.

In addition, the paper highlights the benefits of integrating AI into cyber security. The use of AI-driven defences has been shown to enhance the effectiveness of various mitigation strategies by approximately 25% to 35%, improving overall network security and resilience [27].

The third paper is particularly interesting as it highlights the widespread security vulnerabilities in and around IoT devices. While not a conventional survey, it combines a literature review with a case study, specifically addressing the security of communications within resource-constrained IoT networks. The literature review presents the latest findings on major cyber security threats, including phishing, social engineering, and ransomware, while also examining defence strategies against malware. The case study presents a novel encryption approach specifically designed for IEEE 802.15.4 wireless networks, which are commonly used in IoT devices due to their low power consumption but inherent security limitations. The proposed Genetic Algorithm (GA)-based encryption enhances network security while maintaining optimal performance, effectively balancing protection and efficiency [28].

The last survey provides a comprehensive overview of AI-based ransomware detection, offering valuable insight into the evolving threat landscape. The paper presents compelling data, indicating a 105% surge in ransomware attacks in 2021 alone, with a staggering 232%

increase between 2019 and 2021. Furthermore, the global cost of cybercrime is expected to reach 10.5 trillion by 2025.

A central emphasis of the study is the use of Deep Reinforcement Learning (DRL) for ransomware detection. This method proves to be highly effective in early detection, achieving an impressive accuracy rate of 97%. DRL is particularly beneficial due to its sample efficiency and ability to improve cyber security defences with minimal data requirements. Beyond DRL, the paper also explores alternative detection strategies, including traditional machine learning approaches and hybrid techniques that take advantage of static and dynamic analysis to improve the identification and mitigation of ransomware [10].

3.2 Reinforcement Learning in Cyber Security

When exploring the application of RL in cyber security, most articles concentrate, either completely or partially, on leveraging RL to enhance defensive systems. This is particularly important in light of recent events, such as the AI-powered ransomware attack on an Indian hospital in 2024 [29]. Nonetheless, a smaller body of research is also beginning to explore the potential of RL for offensive strategies, providing valuable insights into its application in executing cyber attacks.

Contemporary research frequently centres on techniques aimed at evading malware detection systems. These studies often employ RL models and address adversarial attacks, in which malicious files are manipulated in specific ways to bypass detection mechanisms [30–33].

Among these studies, a notable contribution is the Evolutionary Generative Adversarial Networks (EGAN) framework, which introduces a novel approach to the generation of adversarial ransomware. This framework uses evolutionary strategies and generative adversarial networks to craft adversarial ransomware capable of evading detection by both AI-powered and traditional antivirus solutions. While preserving the original functionality of ransomware samples, EGAN optimises attack actions and demonstrates a high evasion rate against commercial antivirus software. Beyond its offensive potential, EGAN may also support defensive strategies by generating adversarial samples that can be used to improve detection systems through adversarial training. Nevertheless, the framework presents certain limitations, including its reliance on black-box classifiers, which narrows the scope of evasion. Furthermore, its mutation strategies may lack robustness against dynamic analysis techniques used by some antivirus solutions, and the computational resources required to generate adversarial malware can be intensive [31].

Another significant study in adversarial malware research explores the application of DRL to create adversarial PDF files capable of bypassing detection systems while leaving the documents intact. The paper presents an attack framework based on Double Deep Q-Network architecture, allowing an AI agent to learn how to subtly modify PDF files to evade security systems. In addition to this offensive strategy, the study also presents a defence method that employs RL to revert altered PDFs to their original state, thereby improving detection accuracy. Experimental results demonstrate effectiveness of both the

attack and defence strategies, successfully evading detection while improving the system's ability to recognise genuine files. A key advantage of this approach is its innovative use of DRL, which not only provides a new perspective on evasion attacks but has also shown to be highly effective against existing detection systems. Nonetheless, DRL techniques are complex and require significant computing power, making them difficult to implement. In addition, the success of attack and defence strategies heavily depends on the quality of training data, which could potentially limit real-world applicability [32].

Lastly, an additional line of research explores the application of RL to generate evasive Windows Portable Executable (PE) malware capable of evading static machine learning-based detection models. One of the primary advantages of this approach is its ability to perform black-box attacks, eliminating the need for access of the inner workings of the detection model. The proposed framework maintains the operational integrity of the modified malware and has been released as an open source on the OpenAI Gym platform, enabling further research. This method effectively automates malware evasion, helps identify weaknesses in detection systems, and could potentially be applied to other malware formats. Nevertheless, its success depends on the quality of the dataset and may not outperform methods that exploit more model-specific information [30].

RL is not only employed to enhance attack methods but also plays a crucial role in planning attacks and executing intrusions in preparation for a broader offensive. An article published in 2021 identified four main types of cyber attacks in which RL is leveraged for malicious purposes. These attack types include [34]:

- **False Data Injection (FDI):** A cyber attack in which malicious actors insert incorrect data into a system to manipulate decision-making processes or disrupt operations. RL is employed to more accurately mimic normal system behaviour, allowing the attack to remain undetected for extended periods.
- **Denial of Service (DoS):** An attack in which a system, network or service is flooded with an excessive amount of traffic or requests, rendering it inaccessible to legitimate users. RL is applied to optimise resource allocation, enabling attackers to maximise the impact of the attack despite resource limitations.
- **Cyber-Topology Attacks:** This type of attacks target a network's structural design, manipulating or disrupting its topology (connections between nodes) to degrade performance or cause system failures. RL enables attackers to identify the most impactful action among multiple possibilities while considering resource constraints.
- **Data Poisoning Attacks:** A method in which attackers corrupt the training data of AI or ML models to reduce their accuracy or alter their behaviour.

While attacks on ML classification models, networks, or web systems can have serious consequences, cyber attacks on cyber-physical systems, such as smart grids or autonomous vehicles, pose far greater risk and can lead to catastrophic outcomes.

Researchers have demonstrated that sequential topology attacks can cause cascading failures in power grids. RL-based techniques, such as Q-learning, enable attackers to identify

the most vulnerable sequences of failures. Moreover, RL enhances the effectiveness of FDI attacks by allowing attackers to manipulate sensor readings, thereby disrupting voltage regulation and compromising overall system stability.

By optimising attack sequences, identifying vulnerabilities and increasing the efficiency of adversary strategies, RL substantially amplifies the impact of cyber attacks on cyber-physical systems [34].

3.3 Summary

In conclusion, Table 3.1 offers a concise summary of the literature reviewed, providing a clear overview of all the examined studies. Additionally, the second section outlines the previous work that underpins this thesis. The table highlights four key aspects that are addressed in the majority of the analysed studies.

- **AI-Method:** Specifies the AI technique employed in the malware, e.g., RL, EGAN or DRL.
- **Adversarial:** Indicates if the AI technique is used specifically to create an adversarial attack that targets and evades an AI-based security system. In this case, the AI technique modifies malware to mislead or evade the ML model designed for detection.
- **Evasion:** Describes the type of defence mechanism the malware is attempting to evade. This may involve static detection, where the malware is analysed without being executed, or dynamic detection, where the malware is run in a controlled environment (e.g. a sandbox) to observe its behaviour.
- **Phase:** Refers to the specific point at which the AI technique is applied, either prior to the attack, to prepare or optimise the malware, or during the attack, as it actively seeks to evade detection.

Table 3.1 reveals that the majority of the studies adopt a static approach to malware analysis. For instance, the study by Merlin et al. [33], which builds upon earlier work by Kolosnjaji et al. [30], employs both DQL and the REINFORCE algorithm to modify Windows PE files for evasion. A key finding from this research was that the REINFORCE algorithm outperformed DQL in several scenarios. Intuitively, one might expect a more sophisticated algorithm such as DQL to produce better training results than a simpler approach such as REINFORCE. However, due to its policy-based nature, REINFORCE demonstrated superior learning efficiency and accuracy compared to DQL. This underscores the importance of evaluating a wide range of RL algorithms, rather than relying solely on the latest or most advanced techniques.

Despite these insights, there is a considerable knowledge gap remains regarding the application of RL techniques to train ransomware for evasion, particularly in the context of dynamic malware analysis.

Table 3.1: Review and Comparison of Related Literature

Reference	Year	Survey	AI-Method	Algorithm	Adversarial	Evasion	Phase
[9]	2020	yes					
[27]	2024	yes					
[10]	2024	yes					
[34]	2021	yes	RL				
[26]	2023	no	RL				
[28]	2024	no	GA	GA	no	-	before
[30]	2018	no	RL	DQL	yes	static	before
[31]	2023	no	EGAN	ES & GAN	yes	static	before
[32]	2023	no	RL	DDQL	yes	static	before
[33]	2022	no	RL	DQL & RF	yes	static	before
[6]	2023	no	RL	DQL	yes	dynamic	during
[7]	2024	no	RL	DQL	yes	dynamic	during
This work	2025	no	RL	DDQL & PPO	yes	dynamic	during

Furthermore, most of the algorithms used in existing studies rely on value-based algorithms, with DQL being the predominant choice. This is also the algorithm employed in the prior research that forms the foundation of this thesis [6]. However, there is limited exploration into whether policy-based or actor-critical methods could be more effective in training malware. This thesis seeks to address this gap by implementing a novel RL-powered malware and comparing its performance against the outcomes of the aforementioned prior study.

Chapter 4

Scenario

This chapter outlines the central concept and key components of this work. Following a brief introduction, a comparative analysis of different RL algorithms is conducted in order to determine the most suitable approach for the given context. The selected algorithms are then described in detail, followed by a concise overview of the experimental environment is provided.

The main objective of this research is to explore new RL algorithmic approaches to improve the overall performance and stability of ransomware. In this scenario, the target device is an IoT device subjected to a ransomware attack. Unlike most conventional IoT devices, the target system is equipped with an anomaly detection mechanism capable of identifying unusual behaviour. To evaluate performance, multiple RL algorithms will be tested and compared based on metrics such as encryption speed, detection rate, sample efficiency, and convergence time. The results will be analysed in relation to existing research [8].

4.1 Evaluation of Reinforcement Learning Algorithms

This thesis compares two primary categories of RL methods. The first category consists of RL algorithms that do not employ neural networks (NNs), but instead rely on alternative techniques, such as tabular methods or function approximations, to make decisions. These approaches are typically better suitable to smaller-scale environments with limited state and action spaces, where computational efficiency and simplicity offer key advantages. The objective is to evaluate their effectiveness and potential applicability within this domain.

The second category includes more sophisticated RL methods that use NNs to model complex relationships within the environment. Such algorithms are generally more appropriate for larger and more dynamic environments, as they have a superior capacity to generalise and handle high-dimensional state spaces. By comparing these two types of RL approaches, this study seeks to determine the most effective strategies for optimising ransomware performance, considering key factors such as efficiency, adaptability, and overall success in meeting operational objectives.

4.1.1 Comparison of Basic Reinforcement Learning Algorithms

Basic RL algorithms form the foundation of RL by establishing fundamental strategies for decision-making within dynamic environments. These methods generally do not rely on NNs but instead utilise techniques such as tabular methods, MC and TD learning. They are well suited to problems with discrete and low-dimensional state spaces, where explicit storage and updating of values is computationally feasible.

Despite their relative simplicity, these algorithms play a crucial role in understanding core RL principles such as exploration-exploitation trade-offs, policy evaluation, and value function approximation. This section reviews the fundamental RL methods, comparing their strengths and limitations, while discussing their applicability to different problem settings. The table [4.1](#) provides a comparative overview of these basic RL algorithms, highlighting their main characteristics, advantages, and limitations. It emphasises differences in exploration strategies, convergence behaviour, and model type, providing insight into their suitability for different problem settings explored in the subsequent chapters.

4.1.1.1 State Action Reward State Action

In contrast to Q-learning, whose functionalities were discussed in detail in Section [2.4.6.2](#), State Action Reward State Action (SARSA) does not follow an off-policy learning approach. Instead, SARSA is an on-policy RL algorithm, which means that it updates the action value function based on the current policy. The update step is specified by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \quad (4.1)$$

As shown in equation [4.1](#), the name SARSA derives from the sequence of elements involved in the update process: the current state, the action taken, the resulting reward, the next state and the next action. In contrast to Q-learning, which greedily chooses actions based on the maximum future reward, SARSA updates values according to the action chosen by the current policy, making it more sensitive to the exploration strategy. This results in a more conservative learning process, especially in environments where risky actions should be avoided [\[18\]](#).

4.1.1.2 Linear Function Approximation

Linear Function Approximation shares some similarities with Q-learning, particularly the use of an ϵ -greedy policy for exploration. However, unlike Q-learning, which maintains a table of state-action values, linear function approximation generalises across states using a weighted sum of features. Specifically, the action-value function is represented as a linear combination of state features, with a weight vector that is incrementally updated using gradient-based methods, such as stochastic gradient descent (SGD). This facilitates learning in environments with large or continuous state spaces, where tabular methods such as Q-learning become computationally infeasible [\[18\]](#).

4.1.1.3 REINFORCE

Since the REINFORCE algorithm has already been discussed in Section [2.4.6.3](#), a detailed explanation will not be repeated, however, its main features will be briefly summarised: REINFORCE is a policy-based RL method that optimises an agent's policy using policy gradient techniques. Unlike value-based approaches, such as Q-learning, it directly parametrises and updates a stochastic policy to maximise the expected cumulative reward. The algorithm follows a MC strategy, i.e. updates occur only after an entire episode has been completed. The policy parameters are adjusted using gradient ascent, which reinforces actions that lead to higher rewards while reducing the likelihood of less rewarding actions. Nonetheless, since REINFORCE performs updates based on complete episodes, it is prone to high variance. This issue can be mitigated through techniques such as baseline subtraction, which serve to improve stability [\[18\]](#).

4.1.1.4 Genetic Algorithm

GAs are optimisation methods inspired by the principles of biological evolution and are often employed as an alternative to traditional RL approaches. In contrast to methods that estimate value functions to guide decision-making, GAs operate by evaluating multiple static policies over time. Policies that yield the highest cumulative rewards are selected and randomly mutated or combined to form the next generation, iterating the process to gradually improve performance. These evolutionary methods can be particularly effective when the policy space is small or when the agent lacks complete state information about the environment. However, as they do not learn through direct interaction with the environment, their effectiveness diminishes in cases where real-time feedback could improve decision-making. While evolutionary methods can complement RL, they do not fully exploit the structure of state-action mappings, which often results in slower convergence than learning-based approaches [\[22\]](#).

Algorithm	Type	Model Type	Exploration	Convergence	Advantages	Disadvantages
Q-Learning	Value-based	Model-Free (Off-Policy)	ϵ -greedy, Boltzmann	Can be unstable with function approximation	Mostly converges to optimal policy	May diverge with function approximation, slow in large state spaces
SARSA	Value-based	Model-Free (On-Policy)	Follows policy	More stable than Q-Learning	More stable learning, adapts better to exploration policy	May converge to suboptimal policies
Linear Function Approximation	Value-based	Model-Free (On/Off-Policy)	ϵ -greedy, Boltzmann	More stable than deep NNs	Efficient for large state spaces, interpretable	Limited expressiveness, not suitable for complex problems
REINFORCE	Policy-based	Model-Free	Stochastic policies	High variance, needs many samples	Works well with continuous actions	High variance, slow convergence
Genetic Algorithms	Evolutionary-based	Model-Free	Population-based	No guarantee, depends on mutation/crossover rates	Works in non-differentiable and complex spaces, does not require gradient information	Sample inefficient, slow convergence, no direct reward signal

Table 4.1: Comparison of Basic RL Algorithms

4.1.2 Comparison of Deep Reinforcement Learning Algorithms

The more complex Deep RL algorithms use NNs to handle large, high-dimensional state spaces. In this work, we compare different RL algorithms, selecting one from each major category, with the exception of model-based methods. As discussed in [2.4.6.1](#), these methods require a complete model of the environment, which is not available in this setting.

From the value-based methods, we compare DQL and Double Deep Q-Learning (DDQL), since DQL was the algorithm used in the previous work [\[8\]](#). For policy-based methods, Proximal Policy Optimisation (PPO) is selected, while SAC is chosen to represent hybrid methods. The table [4.2](#) provides a concise comparison of the discussed RL algorithms, highlighting their key aspects and differences. It serves as an overview of the characteristics, strengths, and weaknesses of each algorithm. The following sections provide a more detailed explanation of the algorithms presented in the table, emphasising their unique features and how they differ in terms of performance, stability, and suitability for different types of problems.

4.1.2.1 Double DQL

The primary aim of DDQL is to reduce the overestimation bias, which is a key limitation of DQL. The fundamental difference between DQL and DDQL lies in the way in which the target Q-value is calculated. In DQL, the target Q-value is derived by selecting the maximum Q-value among all possible actions in the next state, which can lead to overestimation. In contrast, DDQL reduces this bias by utilising the primary model to identify the action with the highest Q-value in the next state and then employing the target network to calculate the Q-value for that specific action. Thus, the calculation of the target Q-value in the DDQL algorithm consists of two main steps [\[35\]](#):

1. **Selecting the best action:** Similar to standard DQL, where the action with the highest predicted Q-value is selected, DDQL first selects the action $a_{\max}$ that yields the highest Q-value in the next state s' using the primary network:

$$a_{\max} = \arg \max Q(s', a; \theta) \quad (4.2)$$

2. **Calculating the target Q-value:** Unlike DQL, which uses the same network for both action selection and evaluation, DDQL uses a separate **target network** (θ_{target}) to compute the Q-value for the selected action $a_{\max}$ in the next state:

$$Q_{\text{target}} = r + \gamma \cdot Q(s', a_{\max}; \theta_{\text{target}}) \quad (4.3)$$

Where r is the immediate reward received after taking action a , s' is the next state, and γ is a discount factor that balances immediate and future rewards.

In summary, the DDQL workflow consists of five key steps: (1) initialising the primary and target networks with identical architectures; (2) setting up an experience replay buffer to store the agent’s past interactions, (3) executing the learning loop, in which the agent selects actions using a ε -greedy policy, stores experiences, samples from the buffer, computes target Q-values with both networks, and updates the weights of the primary network; (4) repeating the learning loop until a predefined stopping criterion (such as convergence) is met; and (5) deploying the trained primary network for decision-making. By decoupling action selection and evaluation of the Q-value, DDQL ensures more stable and accurate Q-value estimates, ultimately leading to improved performance in RL tasks [35].

4.1.2.2 Proximal Policy Optimisation

PPO improves basic policy gradient methods by introducing a more stable and efficient way to update the policy. As shown in equation 4.4, it uses a clipping mechanism to limit drastic changes in policy updates, ensuring smoother and more robust learning [22].

$$\mathcal{L}^{\text{PPO-Clip}}(\pi'_\theta) = \mathbb{E}_{\pi_\theta} [\min(\ell_t(\theta') A^{\pi_\theta}(S_t, A_t), \text{clip}(\ell_t(\theta'), 1 - \epsilon, 1 + \epsilon) A^{\pi_\theta}(S_t, A_t))] \quad (4.4)$$

”The function maximises the expected advantage-weighted probability ratio while preventing drastic updates. The expectation is taken over the trajectories generated by the old policy, ensuring that updates are evaluated based on past experiences. The term $\ell_t(\theta')$ represents the probability ratio between the new and old policy, which determines how much the updated policy deviates from the previous one. The advantage function $A^{\pi_\theta}(S_t, A_t)$ quantifies how much better an action is compared to the expected return. The key feature of PPO is the clipping mechanism, which constrains $\ell_t(\theta')$ within $[1 - \epsilon, 1 + \epsilon]$, where ϵ is a small hyperparameter (e.g. 0.2). By taking the minimum between the unclipped and clipped objective, PPO ensures that updates do not push the policy too far in a single step, leading to more stable and reliable learning compared to standard policy gradient methods.”¹

Unlike the Q-learning approaches, which focus on estimating the optimal action value function and selecting actions based on those estimates, PPO directly optimises the policy itself. This direct approach eliminates the need for a value-based action selection process, allowing PPO to handle environments with continuous action spaces more effectively. Its stability has contributed to PPO becoming one of the most popular algorithm in policy-based RL [22].

¹This text was generated using generative artificial intelligence

4.1.2.3 Soft Actor-Critic

As presented in [2.4.6.4](#), the SAC algorithm is an advanced RL method that combines aspects of both policy optimisation and value-based approaches. What distinguishes SAC is its use of maximum entropy RL, which encourages exploration by promoting stochastic policies. Unlike traditional RL methods that focus solely on reward maximisation, SAC simultaneously optimises reward maximisation and policy entropy. This dual-objective framework leads to more stable learning and enhanced performance in continuous action spaces. Maximum entropy aims to ensure that the agent’s policy is not only reward-driven but also encourages randomness, which encourages exploration and prevents premature convergence to suboptimal policies. As a result, SAC offers improved sample efficiency and performs effectively in high-dimensional environments. For these reasons, SAC has become a widely adopted algorithm in modern RL tasks [\[22\]](#).

SAC extends soft policy iteration to incorporate function approximation, thereby enhancing its applicability to real-world problems with continuous action spaces. Rather than directly estimating the true Q-values for policy improvement, SAC performs optimisation on both the value function and the policy. The Q-function is learned by minimising a soft Bellman residual, which integrates the expected payoff with an entropy term. This formulation ensures that the policy remains stochastic and exploratory, a critical feature to avoid overfitting to suboptimal solutions. To further stabilise Q-value estimation, SAC employs two Q-networks alongside their corresponding target networks. The minimum of the two Q-values is used to mitigate the overestimation bias often found in value-based methods, resulting in more stable and reliable Q-value updates.

For policy optimisation, SAC employs the reparameterisation trick in continuous action settings, which result in lower-variance gradient estimates. The policy is implemented as an action network that receives both state and Gaussian noise as input. This method reduces variance compared to alternative approaches, such as likelihood ratio estimators, thereby improving the efficiency and stability of the optimisation process. In addition, SAC dynamically adjusts the entropy regularisation coefficient, α , to ensure that the entropy of the policy remains close to a target value, κ . This automatic adjustment helps maintain the right balance between exploration and exploitation, preventing the policy from becoming excessively deterministic or overly random over time. By allowing such dynamic adjustments, SAC can continue to perform robustly in high-dimensional continuous action spaces throughout the learning process.

Algorithm	DQL	DDQL	PPO	SAC
Model Type	Model-Free	Model-Free	Model-Free	Model-Free
Policy Type	Off-Policy	Off-Policy	On-Policy	Off-Policy
Value Approximation	DNN	DNN	DNN	DNN
Temporal Difference	Yes	Yes	No (Policy Optimization)	Yes
Exploration Strategy	Epsilon-Greedy	Epsilon-Greedy	Clipped Policy Updates	Entropy Regularization
Stability	Higher than Q-Learning	Higher than DQL	High	Very High
Bias and Variance	Overestimation Bias	Reduces Overestimation Bias	Balances Bias-Variance Trade-off	Balances Bias-Variance Trade-off
Scalability	Good (Large State Spaces)	Good (Large State Spaces)	Excellent (Continuous/-Complex)	Excellent (Large and Continuous)
Sample Efficiency	Medium	Medium	High	High
Action Space	Discrete Only	Discrete Only	Discrete and Continuous	Discrete and Continuous
Learning Paradigm	Value-Based	Value-Based	Policy-Based	Actor-Critic (Value + Policy)
Update Method	Q-Value via DNN	Q-Value via DNN	Policy Optimization	Policy and Q-Value Update
Convergence Speed	Good	Good	Very good	Very good
Training Stability	Stable	More Stable than DQL	Very Stable	Very Stable
Complexity	Medium	Medium	Medium	High
Strengths	Handles High-Dimensional States	Reduces Overestimation Bias	Stability and Robustness	Excellent Exploration and Scalability
Weaknesses	Overestimation Bias	Still Requires Tuning	Computationally Intensive	Computationally Intensive

Table 4.2: Review and Comparison of DRL Algorithms [18, 22, 35–38]

4.1.3 Selection of the Reinforcement Learning Algorithms

As this thesis aims to explore alternative RL methods, its primary objective is to compare the performance of newly implemented RL approaches with the existing results obtained in prior research [8]. In addition, this work aims to explore the limitations of various RL techniques, including testing a simpler method that does not rely on NNs.

Selection of SARSA

Among various model-free RL algorithms, SARSA is chosen over Q-learning, REINFORCE, Linear Function Approximation, and GAs. The main reason for choosing SARSA over Q-learning is its on-policy nature, that is, it updates the policy based on the actions actually taken, rather than the optimal action suggested by the current policy estimate. This feature allows SARSA to incorporate exploration more naturally, potentially leading to a more stable learning process.

REINFORCE was not selected as a policy-based method due to its high variance in gradient estimation, which can slow learning and make convergence less reliable. Similarly, Linear Function Approximation was considered, but ultimately deemed unsuitable for the context addressed in this research due to its limitations in representing complex state-action spaces. Lastly, GAs, while beneficial in some RL scenarios, often require extensive computational resources and are less sample efficient compared to SARSA. Given these considerations, SARSA was identified as a stronger candidate for this thesis.

Selection of DDQL and PPO

The initial implementation of this work employs DDQL, which extends standard DQL by addressing overestimation bias through the use of two separate Q-networks. This modification contributes to greater training stability and improves overall performance in discrete action settings. As prior research has already implemented DQL, the adoption of DDQL allows for a direct comparison while improving both stability and accuracy. By refining an existing method into a more effective version, this research aims to extend the capabilities of RL in the specific context of designing RL-driven malware.

Secondly, PPO was chosen over SAC. Although SAC is a powerful and widely adopted actor-critical method, its high computational demands and complexity render it less suitable for this specific application. Even though SAC excels in continuous action spaces with its entropy-based exploration and sample efficiency, it introduces additional challenges, such as increased hyperparameter tuning and implementation complexity. In light of these factors, PPO was selected as a more practical alternative that allows direct comparison with previous work while maintaining strong performance.

4.1.3.1 Summary

The selection of SARSA, PPO and DDQL is consistent with the research objective of evaluating a wider range of RL methods while maintaining meaningful comparisons with prior work. SARSA provides insight into non-NN based approaches and highlights the

effectiveness of simpler on-policy learning strategies. Meanwhile, DDQL builds on the previous implementation of DQL, mitigating overestimation bias and refining value-based learning in a deep RL setting.

PPO was chosen for its favourable balance between stability and efficiency in policy optimisation. As an on-policy method, PPO offers improved sample efficiency over traditional policy gradient methods, while its clipped objective function ensures stable learning, which prevents excessively large policy updates. This makes PPO particularly well suited for tasks where controlled exploration is critical to avoiding instability. In addition, its widespread use in RL research and practical applications ensures strong comparability and reproducibility.

By analysing the outcomes of these selected methods, this thesis aims to highlight the strengths and weaknesses of different RL approaches within the domain of AI-driven ransomware development, providing valuable insight into the most effective techniques in this context.

4.1.4 Refinements and Extensions

Since the standard offline implementation of DQL includes a replay buffer, an initial extension considered was the integration of Prioritised Experience Replay (PER) [39]. Unlike standard experience replay, where all transitions have the same sample probability, PER assigns higher priorities to transitions with larger TD errors, directing the learning process towards more informative experiences. This approach improves learning efficiency by prioritising transitions that lead to significant policy improvements.

However, the original work by [8] did not implement a replay buffer and used DQL solely in an online setting. In order to maintain comparability with previous results, this study does not investigate PER further and instead follows the original approach of online training without experience replay [22].

4.2 Environment Setup

The environment of this study consists of two main components:

1. Client device (IoT Target) - A Raspberry Pi 3 Model B Rev 1.2 running Electrosense software on Raspbian GNU/Linux 9 (Stretch). The device was equipped with an antenna and a SDR module connected via USB. It was integrated into a Local Area Network via an Ethernet connection.
2. C&C Server - Dell XPS 9720 (i9-12900, 64GB RAM, Win11 Pro) responsible for fingerprint analysis, execution of the RL model, and coordination of attacks.

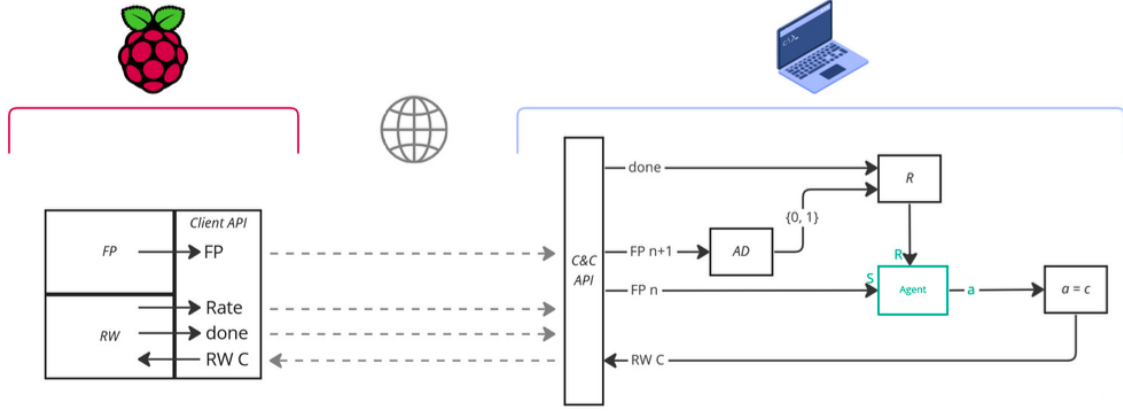


Figure 4.1: Overview of the Client - Server Environment [8]

However, since the primary research objective is to explore novel RL algorithms and their limitations, this study did not reimplement the client-side hardware. Instead, previously collected fingerprint data from earlier work [8] was used. This approach ensures consistent training samples across different RL algorithms, allowing for more reliable comparisons of results.

4.2.1 Ransomware

In this thesis, a modified Python-based ransomware PoC, originally developed in the underlying work [8], is used to target and encrypt files on the IoT device. The ransomware allows for runtime configuration adjustments, enabling dynamic changes to:

- Encryption algorithm (AES-CTR, Salsa20, ChaCha20)
- Encryption rate
- Burst duration
- Burst pause intervals

It communicates with the C&C server for remote reconfiguration and encryption status reporting. However, the implementation of ransom mechanisms lies beyond the scope of this study.

To ensure comparability with previous research and to evaluate different attack strategies, the same six ransomware configurations from [8] were tested. As shown in Table 4.3, these configurations vary in encryption rates, burst durations, and pause times, and cover the following:

- Low encryption rate (Config 0)
- Aggressive, continuous encryption (Configs 1 & 2 - control group)
- Intermediate encryption strategies with specific burst durations (Configs 3-5)

Table 4.3: Ransomware Configurations [6]

Config	Algorithm	Rate (B/s)	Burst Duration	Burst Pause (s)
0	ChaCha20	16	1 file	60
1	AES-CTR	565,566*	unlimited	0
2	Salsa20	632,835*	unlimited	0
3	AES-CTR	500	10 s	5
4	ChaCha20	200	20 s	40
5	Salsa20	200	120 s	30

*Unlimited by configuration, value computed from average rates in [6].

4.2.2 Fingerprint Data

The system continuously monitored the activity and resource usage, transmitting the collected data to the C&C server every five seconds for further analysis. This approach, originally introduced in previous work [8], was designed to capture detailed behavioural fingerprints while minimising processing overhead on the IoT device.

As aforementioned, this study relies on fingerprint data previously collected from the underlying thesis. This decision ensures consistency across experiments, enhances the comparability between of results, and optimises time efficiency.

Chapter 5

Implementation

This chapter provides a clear and structured overview of the implementation process undertaken in this thesis. ^[1] The first section introduces the foundational RL environment upon which this study is built. It discusses key components, including the design and role of the reward function, various approaches to AD, and the activation functions employed in different RL agents. These elements play a crucial role in shaping the learning process and optimising the performance of the implemented models.

The following sections offer a detailed examination of the specific algorithms implemented in the study, providing insight into their structure, functionality, and observed performance. Each algorithm is introduced with a brief explanation, followed by the results obtained from its execution. It is important to note, however, that this chapter does not directly compare the algorithms. A primary comparative analysis will be conducted in Chapter ^[6], where the various models will be systematically evaluated based on performance metrics, overall effectiveness, and other relevant indicators.

5.1 Reinforcement Learning Environment

Building upon the existing RL environment established in ^[8], this work inherits a foundational framework upon which further improvements were made. This section outlines the main modifications and additions introduced to RL agents, including the design of the reward function for the three selected algorithms, the choice of activation functions for the NNs, and the configuration of the learning rate. These elements are critical in shaping the learning dynamics and impact the overall performance of agents.

¹The implementation of the various algorithms has been refined and improved with the help of artificial intelligence tools (e.g. Chat-GPT or xAI), which have provided optimisations, debugging support and code enhancements.

5.1.1 Anomaly Detection

A major component of the RL environment is the AD module, which evaluates the agent's actions to determine whether they remain hidden or lead to detection. Given that fingerprint data processing has a significant impact on AD performance and variations in AD setup can cause significant differences in outcomes, this work adopts the same AD versions tested in previous research to ensure consistency and comparability.

This study incorporates three types of AD modules. The Simple AD only distinguishes between hidden and detected states. The Advanced (Normal) AD improves this approach by aligning its preprocessing with the RL agent and introducing an advanced preprocessor that removes unstable and highly correlated features, reducing the size of the fingerprint from 103 to 50. This improves classification accuracy for normal behaviour while maintaining effective AD for infected behaviour, ultimately optimising the RL agent's learning process. The reward system assigns +10 to staying hidden with an optimal action, 0 otherwise, and -0.2 for detection.

The last model, called Ideal AD, further refines the system by replacing the AD module's detection mechanism with a manual reward calculation system. This approach eliminates variability in reward determination, ensuring more stable learning and rewards that better approximate expected averages. In addition, the reward computation is now based on the agent's predicted actions rather than the next state, which improves consistency in detection and learning behaviour.

5.1.2 Reward Function

The reward function constitutes a fundamental component of any RL algorithm, as it directly influences the learning process. An inadequately designed reward function can lead to issues such as reward hacking, where the agent identifies unintended strategies that maximise rewards without genuinely achieving the intended objective. Additionally, sparse or misleading rewards, where feedback is infrequent or signals that are poorly aligned with desired outcomes, can result in slow convergence, the development of suboptimal strategies, or even a complete failure to learn the desired behaviour. Such deficiencies may ultimately lead to suboptimal learning performance, unintended agent behaviour, and inefficient training dynamics.

In previous work [8], the following reward function was evaluated and identified as the best option:

$$\textit{Hidden}: 10 \cdot \ln(r + 1) + h \quad (5.1)$$

$$\textit{Detected}: \frac{-d}{\max(r, 1)} - d \quad (5.2)$$

The following parameter values have been used: $h = 0$, $d = 20$.

However, a major limitation of this reward function lies in its inability to account for encryption speed. In the work of [7], this limitation was addressed and adjusted by introducing a new variable s . The final reward function used in this study is as follows:

$$\text{Hidden: } s \cdot \ln\left(\frac{r}{s} + 1\right) + h \quad (5.3)$$

$$\text{Detected: } \frac{-d}{\max(r, 1)} - d \quad (5.4)$$

The following parameter values have been used: $s = 100$, $h = 0$, $d = 20$.

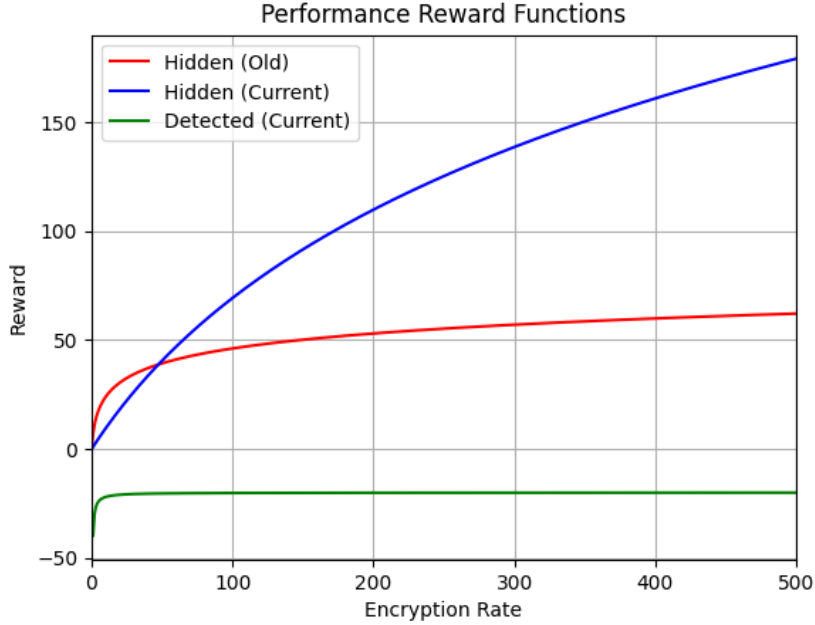


Figure 5.1: Overview of the reward functions

In order to ensure maximum comparability with previous work while incorporating encryption speed into the reward function, the same reward functions as in [5.3] and [5.4] were adopted for this study. The progression of the rewards can be seen in the figure [5.1].

5.1.3 Agent Functionalities

This thesis explores a wide range of approaches by extending and reinterpreting elements of the existing implementation. The DDQL algorithm builds directly on previous work, enhancing the original framework, whereas PPO introduces a novel RL approach. In addition, this research revisits fundamental principles by implementing a tabular SARSA algorithm, which demonstrates a basic RL method that works without a NN.

5.1.3.1 Activation Function

In order to introduce non-linearity, which enables the deep networks to learn complex patterns, an activation function is incorporated. In the DDQL implementation, the NN architecture, adapted from previous work utilises a primary logistic activation function, followed by a secondary Sigmoid Linear Unit (SiLU) activation function. Although a Rectified linear unit (ReLU) offers a simpler and more computationally efficient alternative, it suffers from the well-documented ReLU dying problem, where certain neurons become inactive and cease to contribute to learning.

The PPO implementation addresses the "dying ReLU" problem by adopting the Leaky ReLU activation function, an improved version of ReLU, which avoids this problem by allowing a small, non-zero gradient when the input is negative. This adjustment ensures continuous gradient flow and preserves neurons activity throughout the training process, even when receiving negative inputs.

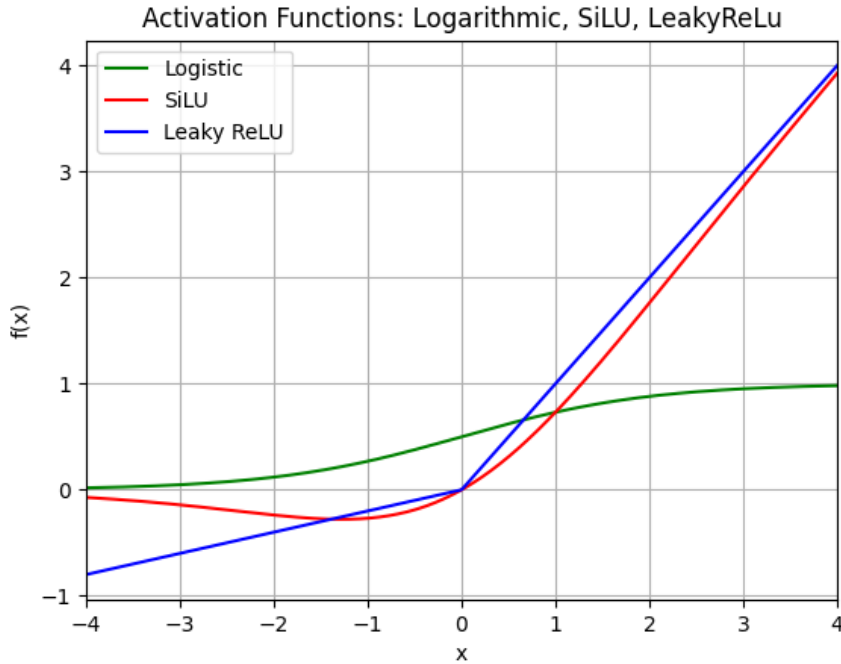


Figure 5.2: Overview of the different activation functions

Unlike the aforementioned algorithms, the SARSA implementation does not incorporate an activation function. As a tabular method rather than a NN-based approach, SARSA does not require an activation function. Instead of approximating Q-values using a function, the algorithm updates Q-values directly within a table.

5.1.4 Processed Input Data

Prior to initiating the first training phase, input data from the IoT device were collected. The behavioural device fingerprints, representing the activity of the device under different

conditions, were stored as CSV files on the C&C server. These datasets included both normal behaviour and compromised behaviour after ransomware execution, providing a comprehensive basis for subsequent processing and model training.

Each fingerprint consists of 103 features that represent various system resource trackers and hardware event counters. While some features are stored as floating point numbers with a theoretically infinite range, the majority are integers. Their values can range widely, from zero to several hundred thousand, depending on system activity. Direct input of raw values into RL models, such as DDQL, PPO, or SARSA, can lead to unstable training dynamics. In value-based methods like DQL and SARSA, large input values can result in excessively high Q-values; when multiplied by network weights, this may cause exploding gradients and hinder learning convergence. Even in policy-based methods such as PPO, unscaled input distributions may disrupt policy updates, leading to inefficient optimisation.

To overcome these challenges, the input data require appropriate preprocessing. Although normalisation, which scales values to a fixed range such as $[-1, 1]$, is a common approach, it is often impractical in real-world RL scenarios. This limitation arises because normalisation requires prior knowledge of the full data set, which is only feasible in controlled or simulated environments where all possible input variations (e.g. fingerprints) are known beforehand. In more dynamic or unpredictable settings, standardisation is often used instead, as it relies on statistical properties such as mean and standard deviation calculated from observed data, making it more adaptable to varying input distributions.

In our case, we apply Min-Max scaling using bounds derived from the training distribution. This provides a consistent preprocessing strategy while still adapting to the practical constraints of real-world data collection.

$$X' = \frac{X - \min_d}{\max_d - \min_d} \quad (5.5)$$

This transformation ensures that all input features are placed on a consistent scale, which is essential for stabilising NN training and improving convergence.

Nevertheless, Min-Max scaling has certain limitations. Since the values $\min_d$ and $\max_d$ are fixed based on the training data, newly collected fingerprints during inference or testing may fall outside this predefined range. As a result, scaled values may extend beyond the $[0, 1]$ interval. Despite this drawback, Min-Max scaling remains a practical and widely used technique, especially when it is necessary to constrain feature values within a specific range, as extreme inputs can significantly disrupt the learning process.

By structuring the collected data accordingly and applying appropriate preprocessing techniques, the system ensures that the learning agents receive well-conditioned inputs, ultimately improving their ability to detect and respond effectively to ransomware threats.

5.2 Prototypes

As discussed in section [5.1.3](#), this work implements three RL algorithms: DDQL, tabular SARSA, and PPO. In order to evaluate their effectiveness, two prototype configurations have been developed for each approach: one incorporating normal AD behaviour and the other using ideal AD behaviour. The following section presents these prototypes in detail, highlighting their individual performance and characteristics. It is essential to understand the standalone behaviour of each algorithm prior to the comparative analysis, as it provides deeper insight into their respective strengths, weaknesses, and overall suitability for ransomware detection and mitigation.

5.2.1 DDQL Prototype

Prototype Version	Neural Network Structure	Anomaly Detection
v20 DDQL	3 Layers (1 Hidden layer)	Normal AD
v21 DDQL	4 Layers (2 Hidden layer)	Ideal AD

Table 5.1: DDQL – Overview of Prototype Differences

Versions 20 and 21 introduce the new DDQL implementation. Version 20 incorporates the advanced AD method and Version 21 adopts the ideal AD approach, as well as an additional layer in the NN. The architecture of the DDQL algorithm consists of two separate NNs: a primary model, responsible for making predictions and learning, as well as a target model, which supports consistent learning by providing a reference point. The primary NN architecture consists of an input layer, one or two hidden layers (as seen in **v21**), and an output layer. The size of the input layer is determined by the number of preprocessed fingerprint features, while the output layer corresponds to the number of possible actions.

In **v21**, the first hidden layer contains twice as many units as the number of fingerprints from the input (100 nodes), while the second contains the same number as the input (50 nodes). In contrast, **v20** employs a single hidden layer with an optimal size of 80% of the input dimension, rounded to the nearest multiple of 5, resulting in 40 nodes. As outlined in section [5.1.3.1](#), the DDQL uses the sigmoid function in the hidden layers to introduce non-linear transformations by mapping inputs to a range between 0 and 1. The output layer employs the SiLU activation function, which blends linear and sigmoid characteristics to generate Q-values effectively. The weights of the target model are then periodically updated by copying the weights of the primary model. Weight initialisation follows the He method, originally developed for ReLU-based activations, but adapted in this case to support stable learning. Optimisation is performed using custom backpropagation with a decaying learning rate starting at $\alpha = 0.005$ and gradually decreasing over training episodes. An epsilon-greedy exploration strategy (with an initial $\epsilon = 0.4$) is used to balance exploration (trying new actions) and exploitation (selecting actions with the highest predicted value) during training.

Algorithm 5.1 Double Deep Q-Learning (DDQL)

```

1: Initialize: Learning rate  $\alpha$ , discount factor  $\gamma$ , exploration rate  $\varepsilon$ , primary NN weights  $W_p$ , target NN weights  $W_t \leftarrow W_p$ 
2: for each episode do
3:   Initialize state  $S$ 
4:   for each step do
5:     Select  $A$  using  $\varepsilon$ -greedy on  $Q_{W_p}(S, A)$ , take  $A$ , observe  $R, S'$ 
6:      $Q_{\text{target}} \leftarrow R$  if  $S'$  terminal, else  $R + \gamma Q_{W_t}(S', \arg \max_a Q_{W_p}(S, a))$ 
7:     Update  $W_p$  using loss  $(Q_{\text{target}} - Q_{W_p}(S, A))^2$ 
8:     if  $S'$  is terminal then
9:       break
10:    end if
11:     $S \leftarrow S'$ 
12:  end for
13:  Update  $W_t \leftarrow W_p$ 
14: end for

```

The pseudocode in Algorithm 5.1 outlines the learning process. Each episode begins by resetting the environment and providing the agent with an initial system state. The agent then selects an action by performing a forward pass through its network to predict Q-values, choosing either a random action for exploration or the action with the highest predicted value for exploitation. After executing the action, the environment returns a new state, a reward, and a flag indicating whether the episode has ended. For each time step, the agent performs another forward pass through the target network to obtain stable Q-values, followed by a backward pass to update its network weights, resulting in three computational passes per step. Upon completion of the episode, the target network is updated to match the main primary, and a new episode begins.

5.2.1.1 Results

Prototype	Episodes	Accuracy (in %)	Training Time (in min)
v20 DDQL	200	88.86	3.8
v20 DDQL	300	91.50	5.7
v20 DDQL	1000	97.04	20.7
v20 DDQL	10'000	99.64	209.5
v21 DDQL	200	89.82	4.0
v21 DDQL	300	92.57	6.5
v21 DDQL	1000	97.00	22.1
v21 DDQL	10'000	99.75	226.5

Table 5.2: DDQL - Training performance across different episode lengths

To summarise the performance of the two new algorithms, both reach an accuracy of about 90% shortly after 200 episodes, requiring around 5 minutes of training. As training

progresses, both prototypes show continued improvement, with **v21** initially outperforming **v20** in terms of accuracy due to the additional hidden layer, which enhances its representational capacity and learning stability.

Specifically, after 300 episodes, **v21** achieves an accuracy of 92.57%, compared to 91.50% for **v20**, indicating a slight yet significant performance gain. This improvement, however, comes at the expense of longer training time and increased computational complexity for **v21**, as elaborated in Chapter 6.1. With extended training periods, the performance gap becomes more pronounced, with **v21** reaching an accuracy of 99.75% after 10,000 episodes. These findings highlight the trade-off between training time and performance, while also emphasising the benefits of deeper neural architectures, provided their higher computational requirements are accounted for.

To further assess the capabilities of both configurations, **v20** and **v21** were trained with different episode lengths to evaluate their performance in terms of speed, convergence, average reward, and stability. The results suggest that the DDQL algorithm **v21** demonstrates a slightly better performance due to the additional NN. A progressive learning curve can be seen in figure 5.3, particularly in the first and second plots, which illustrate progression in both instantaneous and average rewards per episode.

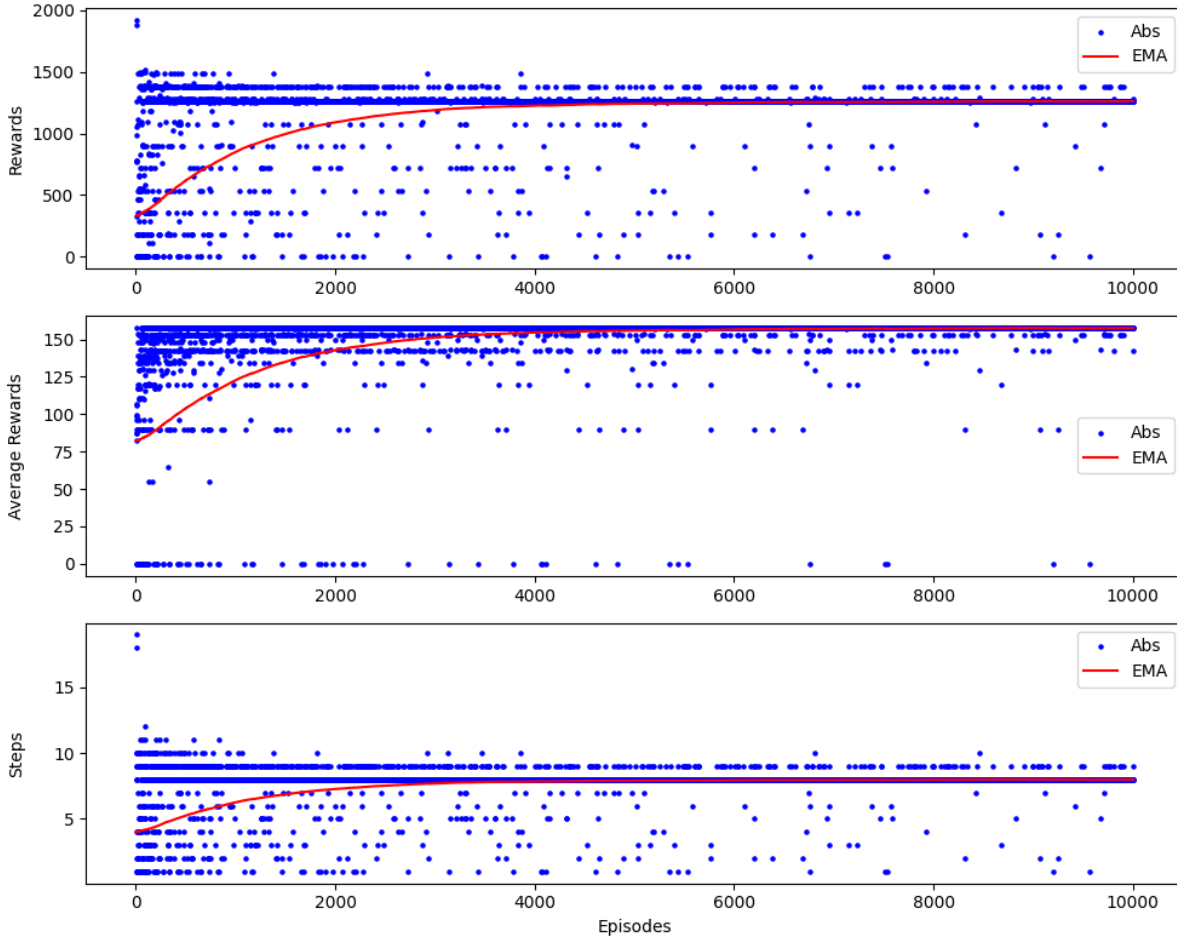


Figure 5.3: Performance Results of **v21** after 10,000 Episodes

The third plot tracks the number of steps taken per episode, revealing a gradual increase, which suggests more stable and prolonged interactions over time. Overall, the results indicate that the algorithm is learning effectively, achieving high rewards while maintaining stability.

Configuration	Initial Accuracy	Trained Accuracy	Final Q-Values
	Percentage #Samples	Percentage #Samples	
Config 0	07.04% (197/2800)	00.04% (1/2800)	15.18312278
Config 1	06.93% (194/2800)	00.04% (1/2800)	-0.18787668
Config 2	50.82% (1423/2800)	00.07% (2/2800)	-0.25345704
Config 3	21.79% (610/2800)	99.75% (2793/2800)	159.07990682
Config 4	05.82% (163/2800)	00.07% (2/2800)	110.20552586
Config 5	07.61% (213/2800)	00.04% (1/2800)	110.20156486

Table 5.3: Accuracy results for Prototype 21 after 10,000 episodes $\epsilon = 0.4$, $\delta = 0.01$, $\alpha = 0.005$, $\gamma = 0.9$

After 10,000 training episodes, the accuracy outcomes across the different ransomware configurations show a clear variation. Initially, Config 2 demonstrated the highest accuracy at 50.82%. However, as training progressed, the agents consistently prioritised Config 3, which proved to be the most effective, achieving a near-perfect accuracy of 99.75%. This is further supported by the final Q-values, with Config 3 achieving the highest score of 159.09, indicating its superiority in the learnt policy. In contrast, the remaining configurations showed a significant decline in accuracy, confirming that the agent reliably identified and converged on the optimal configuration.

5.2.2 SARSA Prototype

The SARSA algorithms aim to optimise ransomware detection through a structured learning process. As previously mentioned, they do not rely on NNs, but instead employ a tabular approach.

Prototype Version	Function Representation	Anomaly Detection	Discretisation
v22 SARSA	Tabular (Q-Table)	Normal AD	One decimal
v23 SARSA	Tabular (Q-Table)	Ideal AD	Two decimals

Table 5.4: SARSA – Overview of Prototype Differences

To evaluate the limitations of this method, two prototypes with different strategies were developed. **v22** adopts a simpler design, featuring minimal state preprocessing, no standardisation and discretisation to a single decimal place. It adopts conservative learning

parameters (learning rate of $\alpha = 0.05$, discount factor of $\gamma = 0.9$) alongside a faster exploration decay of $\delta = 0.01$. The agent is trained using a state-based reward system within an earlier version of the environment.

Algorithm 5.2 Pseudocode for Tabular SARSA

```

1: Initialize: Learning rate  $\alpha \in (0, 1]$ , Discount factor  $\gamma \in [0, 1]$ , Exploration rate  $\varepsilon > 0$ ,
   Decay rate for  $\varepsilon$ , Q-table =  $\{\}$ 
2: Load feature indices from the preprocessed dataset
3: for each episode do
4:   Adjust  $\varepsilon$  using the decay rate
5:   Observe initial fingerprint, pre-process and discretize it to get initial state  $S$ 
6:   Choose action  $A$  using an  $\varepsilon$ -greedy policy ▷ e.g.,  $\varepsilon$ -greedy
7:   while episode is not finished do
8:     Take action  $A$ , receive reward  $R$ , and observe new fingerprint
9:     Process and discretize fingerprint to get next state  $S'$ 
10:    if  $S'$  is new in Q-table then
11:      Initialize Q-values for all actions at  $S'$ 
12:    end if
13:    Choose next action  $A'$  using  $\varepsilon$ -greedy policy ▷ e.g.,  $\varepsilon$ -greedy
14:    Update Q-value:
        
$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$$

15:    Move to the next state:  $S \leftarrow S', A \leftarrow A'$ 
16:  end while
17:  Save updated Q-table
18: end for

```

In contrast, **v23** takes a more advanced approach, incorporating feature standardisation and finer discretisation (two decimal places). It adopts the same learning parameters as **v22** with a learning rate of $\alpha = 0.05$ and a discount factor of $\gamma = 0.9$ as well as an exploration decay of $\delta = 0.01$. Rather than employing a state-based reward system, it uses an action-based model tailored to a more advanced environment. These refinements aim to improve the handling of complex state spaces while maintaining a tabular structure.

Both implementations discretise continuous fingerprints by rounding values to one or two decimal places, thereby simplifying the state space. These discrete states are mapped to Q-values in a Q-table, which is updated using the SARSA rule, based on the current action, the next chosen action, and the received reward (e.g. +10 for remaining undetected, -0.2 otherwise).

Training proceeds over multiple episodes, during which the agent performs state transitions, action predictions, and Q-value updates. The Q-table is stored in a JSON file to ensure persistence, allowing the agent to retain its acquired knowledge for future use. This method balances memory efficiency with adaptability, initially assigning zero values for new states and gradually updating them based on experience.

5.2.2.1 Results

The performance of the SARSA agent clearly illustrates the limitations of tabular RL in high-dimensional environments. As shown in Table 5.5, the agent exhibits negligible improvement during training, with the final accuracy values remaining close to their initial levels. This outcome suggests that the SARSA algorithm fails to develop an effective policy and largely performs near-random action selection. The primary reason for this underperformance lies in the difficulty of accurately representing and updating Q-values within a large and complex state space. These findings underline the difficulty tabular methods face when applied to environments that require generalisation across large, high-dimensional state representations, highlighting their unsuitability for this type of problem.

Configuration	Initial Accuracy	Trained Accuracy
	Percentage #Samples	Percentage #Samples
Config 0	91.18% (2553/2800)	91.36% (2558/2800)
Config 1	01.57% (44/2800)	01.61% (45/2800)
Config 2	01.96% (55/2800)	01.50% (42/2800)
Config 3	01.71% (48/2800)	01.64% (46/2800)
Config 4	01.68% (47/2800)	01.71% (48/2800)
Config 5	01.89% (53/2800)	02.18% (61/2800)

Table 5.5: Accuracy results for Prototype 23 after 10,000 episodes $\epsilon = 0.4$, $\delta = 0.1$, $\alpha = 0.05$, $\gamma = 0.9$

A major limitation of tabular methods is the vastness of the state space. With only 10,000 episodes, the agent is able to explore only a small fraction of the possible states, leaving most Q-table entries unvisited or infrequently updated. Unlike NNs, which are capable of generalising across similar states, tabular SARSA requires repeated visits to each specific state-action pair to learn meaningful patterns, a highly impractical expectation given the scale of the environment. Even with relatively coarse discretisation (e.g. one decimal), the number of possible states remains prohibitively large. For instance, if only five features have 401 possible discrete values each, the resulting state space already exceeds 10 trillion states, rendering exhaustive exploration impractical. While extending training to millions of episodes could improve coverage, it would require an infeasible amount of computational time.

This challenge is exacerbated by the nature of behavioural fingerprinting, which produces high-dimensional state representations with 50 to 97 features. Many of these features span large theoretical ranges or are continuous, making them difficult to capture effectively using a tabular approach. The results suggest that the SARSA agent’s near-random action selection is not the result of poor policy, but rather a consequence of the fundamental limitations of tabular RL in large, complex, high-dimensional environments.

5.2.3 PPO Prototype

Versions 24 and 25 represent a significant shift, introducing the first policy-based RL algorithm examined in this study: PPO.

Prototype Version	Neural Network Structure	Anomaly Detection
v24 PPO	4 Layers (2 Hidden layers)	Normal AD
v25 PPO	4 Layers (2 Hidden layers)	Ideal AD

Table 5.6: PPO – Overview of Prototype Differences

In contrast to the previously explored value-based methods, DDQL and SARSA, which rely on estimating state-action values to derive a policy, PPO learns a parametrised policy directly. This approach enhances its ability to handle high-dimensional and continuous action spaces as well as improving generalisation in more complex environments. Owing to its stability and efficiency of PPO has become a widely adopted algorithm in modern RL.

Algorithm 5.3 Pseudocode for PPO Agent

```

1: Initialize: Policy and value networks, learning rate  $\alpha$ , discount factor  $\gamma$ , clipping
   parameter  $\epsilon$ , GAE parameter  $\lambda$ , entropy coefficients, optimiser
2: for each episode do
3:   Reset environment and preprocess initial fingerprint to get state  $S$ 
4:   while episode not done do
5:     Sample action  $A$  from policy, get  $\log \pi(A|S)$  and value  $V(S)$ 
6:     Take  $A$ , observe reward  $R$ , new fingerprint, and done flag
7:     Preprocess fingerprint to get new state  $S'$ 
8:     Store  $(S, A, \log \pi(A|S), V(S), R, \text{done})$ 
9:      $S \leftarrow S'$ 
10:  end while
11:  Compute returns and advantages using GAE
12:  Update policy using PPO loss (clipped objective, value loss, entropy bonus)
13:  if evaluation episode then
14:    Evaluate agent and update best model if performance improves
15:    if early stopping condition met then
16:      break
17:    end if
18:  end if
19: end for

```

In the pseudocode shown in [5.3](#), each episode executed by the PPO agent follows a well-structured flow designed for stability and training efficiency. The episode begins by resetting the environment and providing the agent with an initial state fingerprint. At each timestep, the agent samples an action from its current policy, introducing natural randomness without relying on explicit exploration strategies such as ϵ -greedy. After executing the action, the environment responds with a new state, a reward, and a "done" flag

that indicate whether the episode has ended. Unlike value-based methods, the agent does not update its NN in real time. Instead, it stores important information such as states, actions, rewards, logarithmic probabilities, and value estimates. Once the episode is complete, the agent processes the collected data to compute the benefits and returns. These are then used to update the policy and value networks by running multiple optimisation epochs over batches of episode data. Compared to DDQL, the PPO agent requires only a single forward pass per step during the episode, resulting in a significant reduction in computations and a faster episode progress.

As shown in Table [5.6](#), the main difference between v24 and v25 lies in the AD method: v24 uses a standard AD mechanism, while v25 employs an ideal variant. Otherwise, both implementations are identical and share the same set of hyperparameters defined in `settings.py`. These parameters govern the learning behaviour, control the balance between exploration and exploitation, and require careful tuning to ensure optimal performance.

- **Learning rate:** 0.0001 - Determines the step size for the optimiser.
- **Clip Epsilon:** 0.1 - Limits the extend of policy updates to ensure stable learning.
- **Discount Factor (γ):** 0.99 - Controls the weighting of future rewards.
- **GAE Lambda (λ):** 0.95 - Controls the smoothing of benefit estimates.
- **Value Loss Coefficient:** 0.5 - Scales the critic loss in the overall objective.
- **Initial Entropy Coefficient:** 0.5 - Encourages exploration at the beginning of training.
- **Entropy Coefficient Decay:** 0.995 - Gradually reduces exploration over time.
- **Minimum Entropy Coefficient:** 0.0001 - Ensures a minimum level of exploration.
- **Epochs per update:** 5 - Number of iterations over each training batch.
- **Batch Size:** 32 - Number of samples used in each training batch.

In contrast to earlier implementations, which solely relied on NumPy for the NN construction, this PPO implementation employs Keras, a high-level deep learning API that streamlines model design and training, and reduces implementation errors. Both PPO algorithms use a NN with two hidden layers to model the policy and value functions. The input layer corresponds to the number of preprocessed fingerprint features, followed by a first hidden layer with twice as many units, and a second hidden layer with half as many. The network employs leaky ReLU activations and He-normal initialisation to improve training stability and performance.

```

1 def _build_model(self):
2     self.hidden1 = tf.keras.layers.Dense(
3         self.num_hidden1, activation=tf.nn.leaky_relu,
4         kernel_initializer='he_normal'

```

```

5      )
6      self.hidden2 = tf.keras.layers.Dense(
7          self.num_hidden2, activation=tf.nn.leaky_relu,
8          kernel_initializer='he_normal'
9      )
10     self.policy = tf.keras.layers.Dense(
11         self.num_output, activation=None,
12         kernel_initializer='he_normal'
13     )
14     self.value = tf.keras.layers.Dense(
15         1, activation=None,
16         kernel_initializer='he_normal'
17     )

```

Listing 5.1: PPO Neural Network Model

Figure 5.4 illustrates the implementation of the PPO neural network architecture in more detail. The network consists of an input layer (x_i), followed by two hidden layers with He initialisation and Leaky ReLU activation functions ($\mathcal{LR}$). The architecture is divided into two separate heads: a policy head with A linear output neurons (π_i) and a value head with a single linear output neuron (V). In the diagram, $\mathcal{LR}$ denotes a Leaky ReLU activation and $\mathcal{L}$ denotes a linear activation.

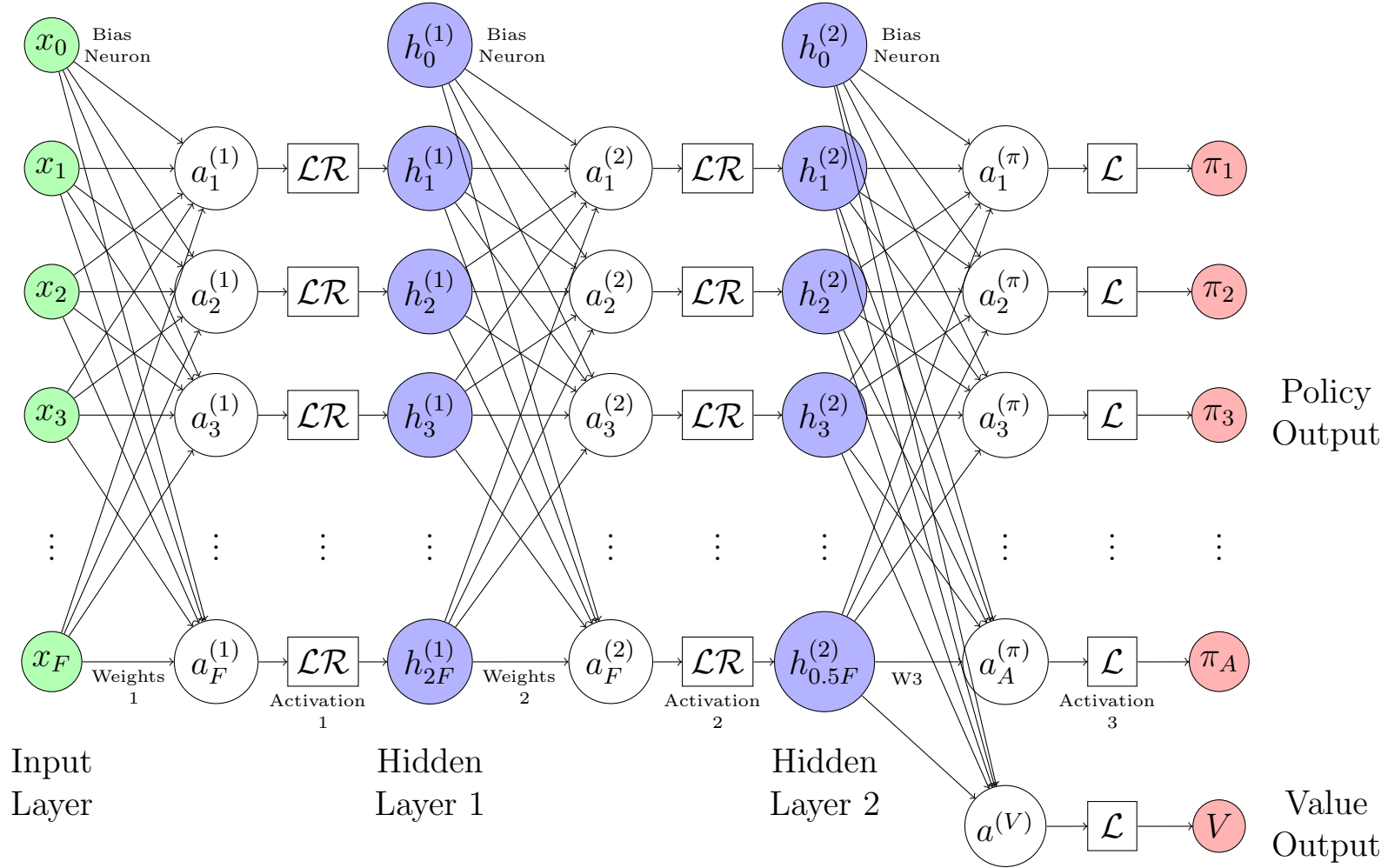


Figure 5.4: Architecture of the neural network for the PPO agent

In the `act` method, the policy network applies a softmax function to generate probabilities for each possible action. An action is then randomly selected based on this distribution, enabling stochastic exploration. Alongside the chosen action, this method also returns its log probability and the estimated value of the current state. This stochastic selection helps the agent explore a variety of possibilities and gradually discover better strategies. The log probability is subsequently employed during training to update the agent's behaviour according to the PPO algorithm.

```

1 def act(self, state):
2     state = tf.convert_to_tensor(state, dtype=tf.float32)
3     if state.shape[0] != self.num_headers:
4         state = tf.pad(state, [[0, self.num_headers - state.shape[0]]],
5                           mode='CONSTANT') if state.shape[0] < self.num_headers else
6                           state[:self.num_headers]
7     state = tf.expand_dims(state, 0)
8
9     policy_logits, value = self.call(state)
10    probs = tf.nn.softmax(policy_logits)[0]
11    action = tf.random.categorical(tf.expand_dims(probs, 0), 1)[0, 0]
12    action = tf.clip_by_value(action, 0, self.num_output - 1)
13    log_prob = tf.math.log(probs[action] + 1e-10)
14    return int(action), float(log_prob), float(value[0])

```

Listing 5.2: PPO action selection method

Other important methods include `update` and `compute_gae`. The `update` method implements PPO's clipped surrogate objective to limit policy changes and maintain training stability. It also computes value loss using a clipping mechanism to limit updates to the value function, and incorporates an entropy term with a decaying coefficient to encourage exploration. Together, these components strike a balance between stable learning and sufficient exploration, enabling the agent to iteratively refine its policies and value estimations. The `compute_gae` method calculates normalised payoffs using Generalised Payoff Estimation (GAE). By combining TD errors with discounting and decay, GAE reduces variance in payoff estimates and improves training efficiency.

5.2.3.1 Results

After tuning the implementation's hyperparameters to identify optimal settings, highly promising results were obtained. The findings clearly demonstrate the advantages of a policy-based approach over value-based methods. Both `v24` and `v25` reached 100% accuracy after only 36 and 44 training steps, respectively. This outcome showcases the efficiency of policy-based learning, which is able to develop an accurate understanding of the environment in significantly fewer training steps compared to value-based approaches.

Figure 5.5 shows the training progress of the PPO agent across 2000 episodes in terms of absolute rewards, average rewards, and number of steps per episode. The blue dots represent individual values, while the red line depicts the exponential moving average, providing a smoothed representation of overall trends.

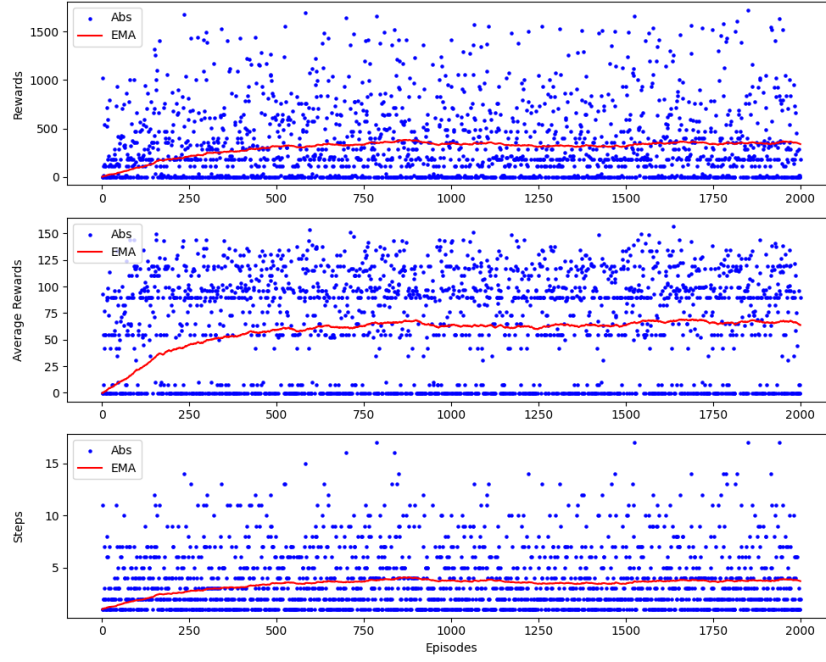


Figure 5.5: Performance Results of v25 after 2,000 Episodes

Although the average reward shows an initial upward trend, there is no clear and consistent convergence to a stable optimum. One reason for this behaviour is the stochastic nature of the PPO algorithm. Unlike value-based methods such as DDQL, which tend to converge deterministically once the optimal policy is identified, PPO maintains a degree of exploration throughout training due to its policy entropy term. This design encourages continuous sampling of alternative policies, even after a near-optimal performance is achieved, leading to variability across episodes.

Prototype	Episodes	Accuracy (in %)	Training Time (in min)
v24 PPO	30	94.00	0.2
v24 PPO	35	95.75	0.2
v24 PPO	36	100	0.2
v24 PPO	1000	100	4.9
v25 PPO	40	94.04	0.2
v25 PPO	44	100	0.3
v25 PPO	50	100	0.3
v25 PPO	1000	100	5.1

Table 5.7: PPO - Training performance across different episode lengths

Overall, the results in figure [5.5](#) demonstrate that PPO can rapidly reach high reward levels. However, due to its stochastic policy nature and continuous exploration, it lacks the smooth and consistent convergence curve seen in value-based methods such as the DDQL algorithm, illustrated in figure [5.3](#).

Chapter 6

Comparison

This chapter presents a detailed comparative analysis of the performance of the three RL algorithms developed in this thesis. The main objectives are twofold: first, to expand the performance analysis of each algorithm, by examining its strengths, weaknesses, and efficiency in the context of ransomware training, while highlighting the differences between their respective versions; second, to directly compare the algorithms to determine which approach proves most effective in this particular use case.

The comparative evaluation focuses on identifying key differences in performance across the three algorithms. It explores aspects such as learning speed, accuracy, sample efficiency, convergence speed, overall reward performance and resource consumption, while also considering the trade-offs inherent in the design and implementation of each approach. These metrics were selected to provide a detailed evaluation of both the operational efficiency and malicious effectiveness of each algorithm. Furthermore, the analysis considers the manner in which each algorithm balances competing goals, such as training efficiency versus reward maximisation, and resource consumption versus optimal performance.

Accuracy is measured through dedicated evaluation scripts. Initially, an untrained agent is created and run against all collected fingerprints, both ransomware configurations and normal behaviour. As expected, its predictions are close to random. The agent is then trained using the current environment parameters. After training, its performance is evaluated again on the same set of fingerprints. This time, the agent demonstrates a significant improvement, indicating informed decision-making rather than random guessing. To prevent biased results, the evaluation employs a separate set of fingerprints that were not used during training. In both evaluation phases, the agent only predicts actions without updating its policy, thus ensuring the integrity of the test data.

Ultimately, this chapter aims to provide a clear understanding of how different RL approaches perform in the context of ransomware development, providing deeper insight into the algorithmic trade-off's and helping to identify which models are better suited for such a high-dimensional, adversarial environment like the one examined in this thesis.

6.1 DDQL Performance differences

As detailed in Chapter 5.2.1, two versions of the DDQL algorithm have been implemented v20 and v21. The main differences between them lies in the architecture of the NN and the AD approach used. Although both algorithms appear almost identical in code, differing by only a few lines, the performance outcomes reveal some notable and significant disparities. These results highlight the sensitivity and critical importance of careful algorithm design.

Accuracy and training speed

As illustrated in Table 5.2, both versions reach a high accuracy of 90% before 300 episodes. v21, which incorporates an additional hidden layer, outperforms v20 in terms of accuracy, especially in the early stages, highlighting the benefits of more complex NN architectures. The accuracy of both versions then improves rapidly, reaching 97% after only 1,000 episodes. From this point on, the convergence to the optimal solution slows slightly, but eventually peaks at 99.64% for v20 and 99.75% for v21 after 10,000 episodes.

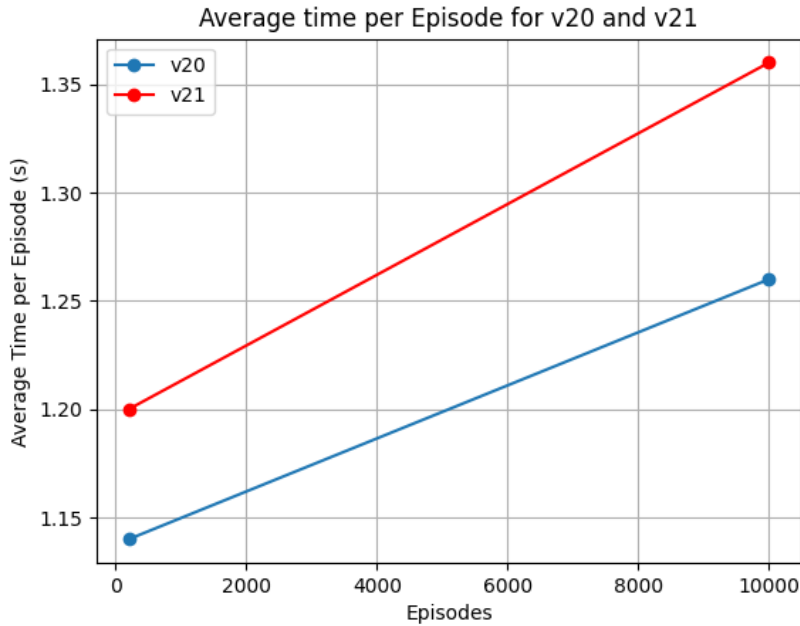


Figure 6.1: Average time needed per episode for v20 and v21

A similar trend is observed in training time. V20, featuring a simpler three-layer architecture (including one hidden layer), completes the first 300 episodes in 5.7 minutes. In contrast, V21, which employs a more complex four-layer architecture (including two hidden layers), requires 6.5 minutes. This is an increase of nearly 15% for the same number of episodes.

Figure 6.1 provides further insight by illustrating the average time per episode. For both algorithms, this average increases as training progresses. It starts at an average of 1.14 and 1.20 seconds per episode for the first 200 episodes and increases to an average of 1.26 and 1.36 seconds per episode by the 10,000th episode. This trend suggests that the AD

system is triggered less frequently in the later stages of training. This is probably because the agent becomes more accurate in choosing effective configurations. As its performance improves, the agent can stay hidden for longer periods of time and more often reaches the maximum episode length, which happened less frequently in the earlier stages.

Resource consumption

A comparison between the CPU and RAM usage reveals that both algorithms exhibit largely similar resource consumption patterns. However, the v21 implementation with the ideal AD approach shows a slightly more stable and consistent resource usage overall, making it more resource efficient compared to v20.

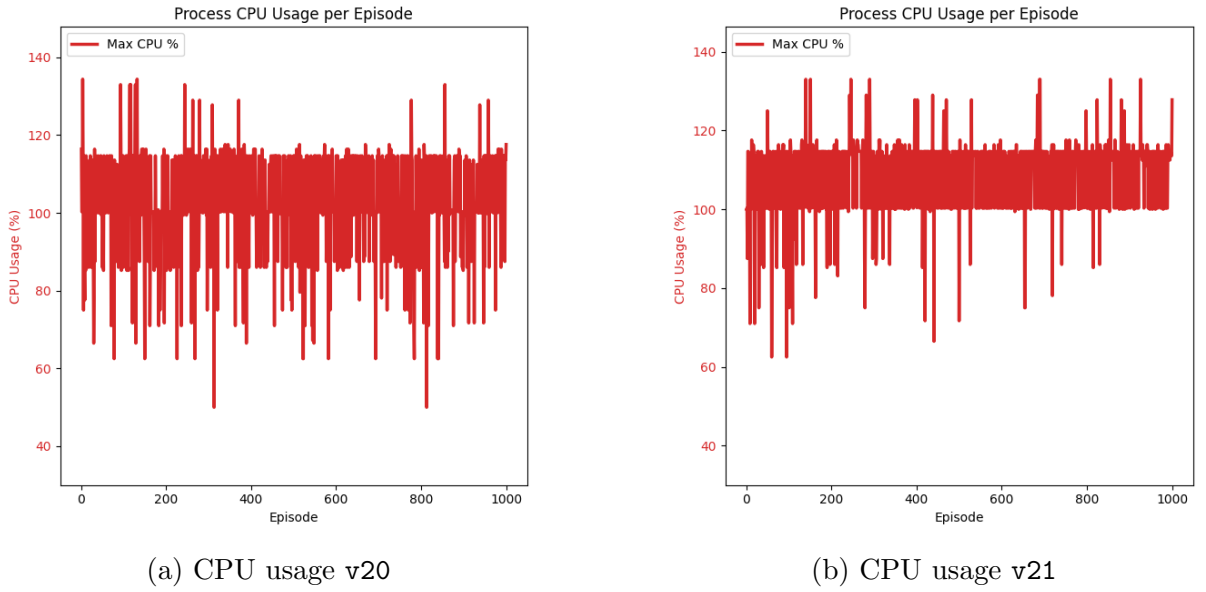


Figure 6.2: CPU usage comparison of DDQL algorithms over 1,000 Episodes

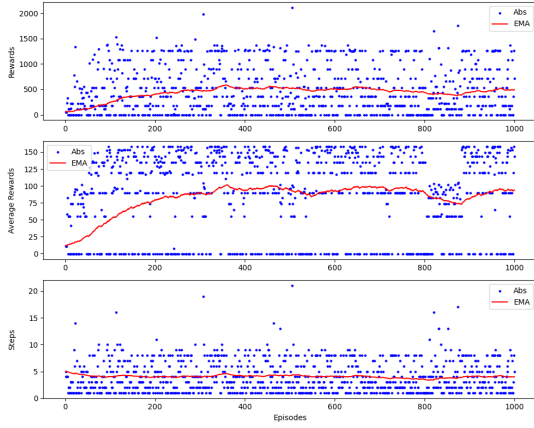
Reward performance

When comparing the average and total rewards of the two DDQL algorithms, Figure 6.3 clearly illustrates how v21 benefits significantly from its modified NN and AD approach. It achieves notably higher total and average rewards compared to v20. In addition, v21 shows a more stable reward behaviour, with a clear convergence toward a final Q-value of 159.1.

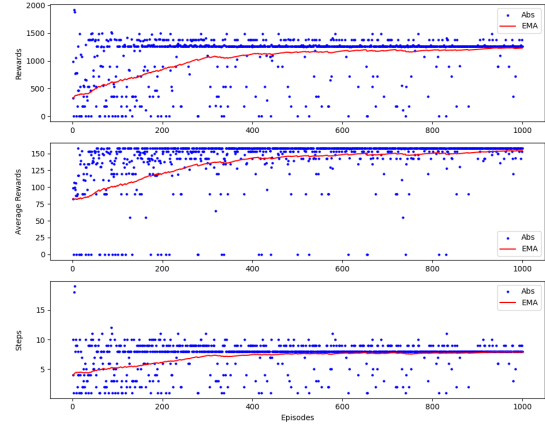
A similar trend is observed in the evolution of the step size. Although v20 begins with a step size of 5, it fails to optimise this value over time, remaining roughly constant without converging to an optimal value. In contrast, v21 successfully converges to a step size of 8.

Other performance indicators

Finally, certain performance metrics, such as sample efficiency, do not demonstrate substantial differences between the two versions. Moreover, there is no absolute threshold that defines when an algorithm is considered sample-efficient. Instead, it is a relative metric, more meaningful when used for direct comparisons, as discussed in Chapter 6.4



(a) Rewards and Step size usage of v20



(b) Rewards and Step size usage of v21

Figure 6.3: Performance analysis of DDQL algorithms over 1,000 Episodes

6.2 Comparison of DQL and DDQL

After 1,000 episodes, v8 (DQL) achieved an accuracy of 96.57% [8], while v21 (DDQL) shows a slight improvement with 97.00%. This indicates a modest but notable increase in accuracy between the two versions.

In terms of training time, DDQL offers a clear advantage by mitigating overestimation bias, leading to more stable learning dynamics and reducing the need for frequent corrective updates. As a result, the overall training duration is significantly shortened. Despite the fact that the DDQL implementation in v21 features an additional NN layer compared to the one used in v8, it completes training in nearly half the time of the DQL implementation. This is evident when comparing the training time after 1,000 episodes: v8 required 42.2 minutes, whereas the DDQL implementation completed training in just 22.1 minutes.

Prototype	Episodes	Accuracy (in %)	Training Time (in min)
v8 DQL	300	91.82	10.8
v8 DQL	1000	96.57	42.2
v8 DQL	10'000	99.72	454.4
v21 DDQL	300	92.57	6.5
v21 DDQL	1000	97.00	22.1
v21 DDQL	10'000	99.75	226.5

Table 6.1: DQL and DDQL - Training performance across different episode lengths

In terms of resource consumption, both algorithms possess a similar structural complexity and require a comparable level of computing power. While DQL holds a slight advantage due to its simpler NN architecture, the differences are minimal and not of practical significance.

In summary, the DDQL approach outperforms the previous DQL algorithm in both accuracy and training efficiency. `v21` attains an accuracy of 97.0% after 1,000 episodes and completes training in just 22.1 minutes, highlighting the clear advantages of DDQL in terms of efficiency, stability and accuracy.

6.3 SARSA with- and without Neural Network

As discussed in Chapter [5.2.2](#), both implemented versions of SARSA face significant limitations in learning the optimal policy due to the extensive size of the state space, which is simply too large for a traditional tabular approach to handle effectively.

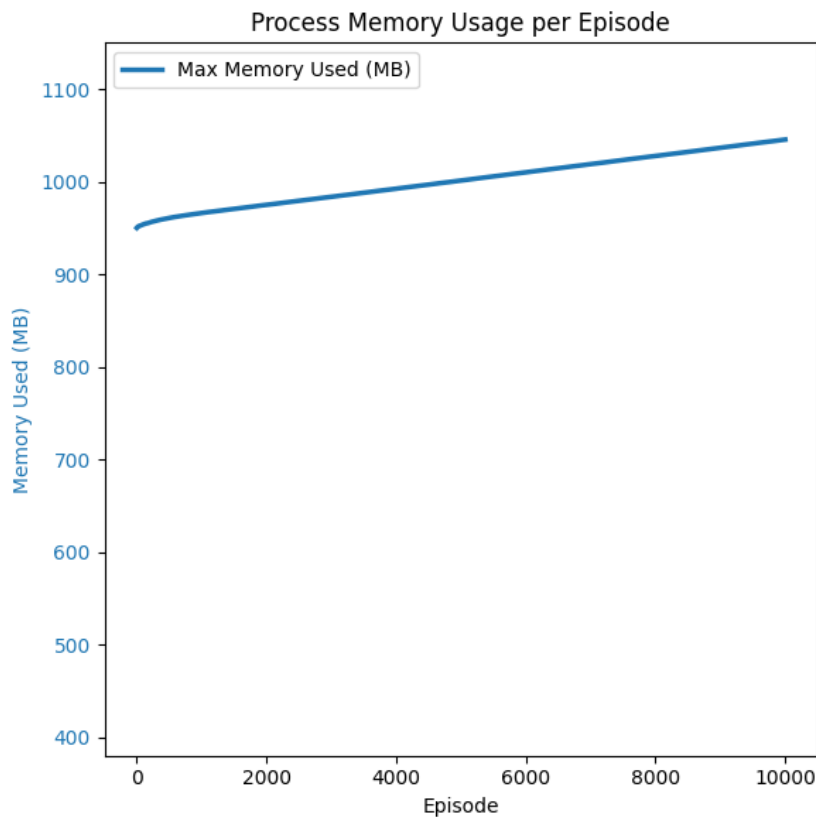


Figure 6.4: Memory usage over 10,000 Episodes of SARSA `v23`

Attempts to improve performance by loading a pre-existing Q-table from a previous run did not yield significant improvements. This is primarily because the preloaded Q-table still lacked comprehensive coverage of all relevant state-action pairs. This limitation highlights a major drawback of the tabular approach in large environments: memory consumption increases rapidly and becomes inefficient. As illustrated in Figures [6.4](#) and [6.6](#) for the DDQL implementation, the tabular SARSA approach consumes nearly twice as much memory. This effectively eliminates one of the main advantages of tabular methods, namely their lightweight nature in small environments.

Another potential strategy to improve performance is to reduce the size of the fingerprint input vector. For example, instead of incorporating 50 features, the input could be limited

to 10 or fewer. However, this simplification would almost certainly degrade the algorithm’s decision-making capabilities. Aggressively reducing the feature set risks the loss of critical information needed to accurately represent the state of the environment, which in turn would affect learning and overall performance.

A more promising alternative is to adapt the SARSA algorithm to a deep variant. This approach was explored in previous work by L  chiger [8], where the deep SARSA model achieved an impressive accuracy of 99.50% after 10,000 episodes, matching the performance to the other algorithms implemented in this thesis. These findings underscore the inherent limitations of basic RL methods that do not incorporate NNs.

6.4 Comparison of DDQL and PPO

A comparison between value-based and policy-based RL algorithms is inherently challenging, as they are built on different paradigms, each characterised by distinct strengths and weaknesses. When comparing DDQL and PPO, particularly the v20 and v24 versions, it is important to acknowledge the unique trade-offs of each algorithm. As a result, evaluating which algorithm performs better is non-trivial, as it depends on multiple factors, including stability, sample efficiency, exploration-exploitation balance, and sensitivity to hyperparameter configurations. Thus, the choice between DDQL and PPO often depends on the specific problem at hand, the available computational resources, and the desired characteristics of the learnt policy.

6.4.1 Accuracy and training speed

When comparing accuracy and training speed, PPO clearly demonstrates the benefits of direct policy optimisation. Its gradient-based updates allow for more efficient learning, resulting in higher performance and greater confidence in the learnt policy with fewer training steps. In contrast, DDQL relies on value estimation and can be more sensitive to overestimation bias and hyperparameter tuning.

As shown in Table 6.2, the performance disparity between the algorithms is clearly apparent. Although the DDQL-based prototypes (v20, v21) slowly converge towards an accuracy ceiling of around 99%, the PPO-based prototypes (v24, v25) consistently achieve 100% accuracy within the first 50 episodes and maintain it throughout the training process.

The high contrast of this data underscores the superior sample efficiency of PPO, as it requires a significantly lower number of interactions with the environment to learn an optimal policy compared to DDQL. Furthermore, table 6.2 highlights the significant differences in training time between DDQL and PPO. For the DDQL implementations (v20 and v21), training 10,000 episodes requires 209.5 and 226.5 minutes respectively, while PPO completes the same task in just 37.8 and 42.9 minutes.

Prototype	Episodes	Accuracy (in %)	Training Time (in min)
v20 DDQL	300	91.50	5.7
v20 DDQL	1000	97.04	20.7
v20 DDQL	10'000	99.64	209.5
v24 PPO	30	94.00	0.2
v24 PPO	1000	100	4.9
v24 PPO	10'000 (6000)	100	37.8
v21 DDQL	300	92.57	6.5
v21 DDQL	1000	97.00	22.1
v21 DDQL	10'000	99.75	226.5
v25 PPO	40	94.04	0.2
v25 PPO	1000	100	5.1
v25 PPO	10'000 (6000)	100	42.9

Table 6.2: Performance results comparison of DDQL and PPO

It is important to note that PPO incorporates an additional automatic stopping mechanism designed to optimise resource usage. After every 1,000 episodes, the model checks for improvements in accuracy; if no enhancements are observed over five consecutive checks, training is stopped early. In practice, this results in PPO completing training after approximately 6,000 episodes. This mechanism not only saves computational resources, but also prevents unnecessary training once the model has converged, improving overall efficiency.

The significant reduction in training time can also be attributed to inherent differences between the two algorithms. DDQL requires multiple network passes per step (two forward passes and one backward pass), resulting in a higher computational load for each episode. In contrast, PPO performs only a single forward pass per step during the data collection phase, with updates occurring in batches at the end of the episode. In addition, PPO samples actions from a learned probability distribution rather than calculating and comparing Q-values for all possible actions, further enhancing efficiency. These structural advantages allow PPO to achieve faster episode progression and more efficient use of computational resources, resulting in dramatically shorter overall training times, even when solving the same task.

6.4.2 Consumption Differences

At first glance, the lower CPU usage observed in **v25** could suggest greater efficiency. However, a more detailed examination of figure [6.5](#) reveals that the DDQL implementation exhibits a more consistent CPU usage, characterised by lower variance and a more balanced load overall.

This suggests that DDQL can use available resources more effectively, contributing to higher computational efficiency. Nevertheless, the PPO algorithm benefits from signif-

icantly higher sample efficiency, enabling it to train the model substantially faster. As a result, despite PPO’s higher variance and less stable resource utilisation, the overall reduction in training time largely compensates for these limitations.

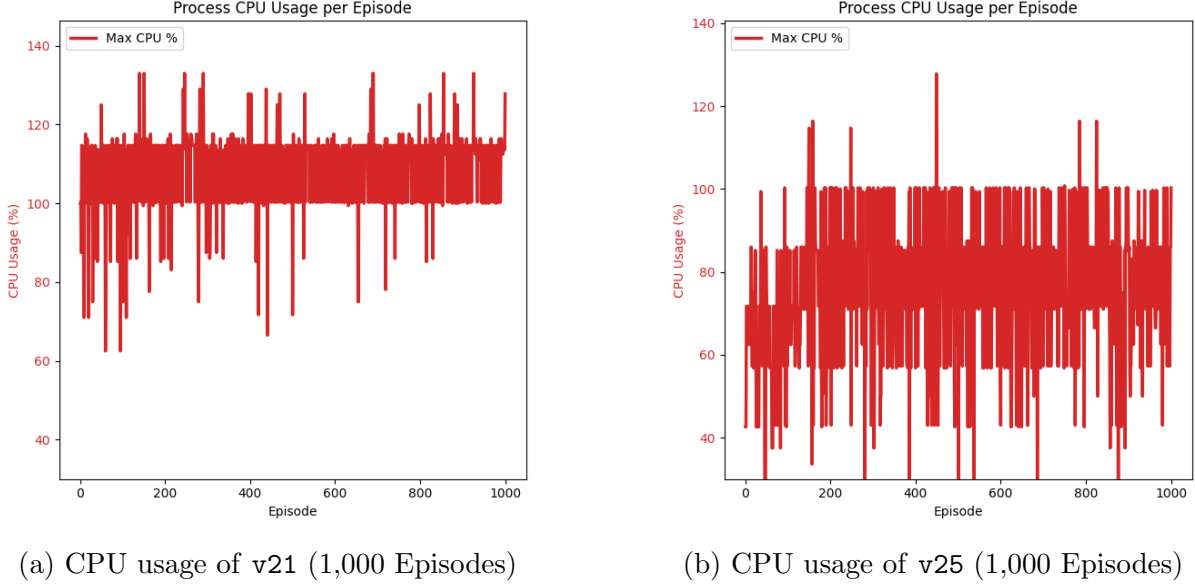


Figure 6.5: Comparison of CPU usage of DDQL and PPO algorithms

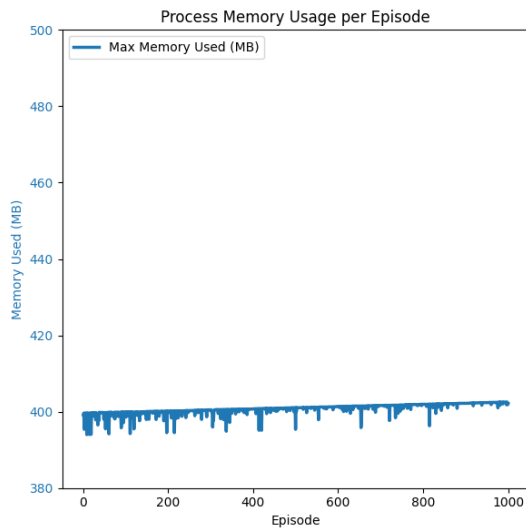
These performance differences align with the theoretical time complexity of each algorithm. For DDQL, the total training complexity can be expressed as $O(E \times T \times n^2)$, where E denotes the number of episodes, T represents the average number of steps per episode, and n is the size of the input feature vector. Each step involves computations that grow quadratically with the input size due to NN operations.

In contrast, PPO has a higher theoretical complexity of $O(E \times K \times T \times n^2)$, where K is the number of optimisation epochs per episode. While this makes PPO about ten times more computationally intensive per episode, it enables the algorithm to learn more effectively from each experience. As shown in Table 6.2, this greater sample efficiency allows PPO to achieve high accuracy, in fewer episodes, which ultimately leads to shorter overall training time and lower energy consumption.

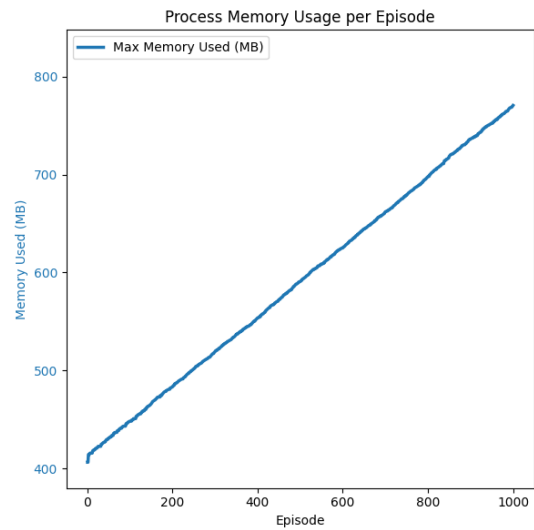
Memory usage presents another key distinction between the two algorithms. As illustrated in Figure 6.6, the memory usage of the DDQL implementation remains relatively stable, with only a slight upward trend over time. In contrast, PPO shows a clear linear increase in memory consumption as training progresses.

This behaviour implies that, depending on the total number of episodes, PPO can end up using significantly more memory, potentially up to ten times more than DDQL in later stages of training. On systems with constrained memory resources (e.g. 4GB), this could pose a practical limitation for training PPO models efficiently.

However, comparable to CPU usage, the primary trade-off concerns the overall training duration. While DDQL is less memory intensive at any given time, it takes significantly longer to reach the same number of episodes. As a result, RAM usage is lower but



(a) Memory usage of v21 (1,000 Episodes)



(b) Memory usage of v25 (1,000 Episodes)

Figure 6.6: Comparison of Memory usage of DDQL and PPO algorithms

sustained over a longer period of time, whereas PPO consumes more memory for a shorter period of time due to its faster convergence. The PPO algorithm is capable of achieving the desired results in less than 50 episodes, which corresponds to a training time of less than one minute. Therefore, it ultimately uses fewer resources and less energy. Although DDQL uses computational resources more efficiently at each point in time, its extended training time results in greater overall resource consumption.

As discussed in section [5.2.3.1](#), despite PPO's great performance in terms of training time and sample efficiency, it struggles to consistently converge to high reward levels, whereas DDQL tends to perform significantly better in this regard.

In summary, both algorithms implemented successfully fulfil the objective of effectively training a ransomware agent and can be considered suitable options, depending on the specific use case. The choice between the two may depend on the underlying hardware used for training and the desired feature extensions. PPO offers clear advantages in terms of higher sample efficiency and easy policy updates. In contrast, DDQL demonstrates greater computational efficiency for the same absolute training time and consistently achieves higher overall rewards, resulting in smoother and more stable convergence.

6.5 Comparisons of additional training Hardware

This chapter examines the impact of hardware setups on the training performance of the implemented RL algorithms. The aim is to assess whether, and to what extent, different hardware configurations affect overall training efficiency.

To ensure fair and consistent evaluation, each of the three algorithms was tested using the version incorporating the ideal AD implementation.

- **Device 1** The first device serves as the primary machine, as described in Section 4.2. It is a Dell XPS 9720 equipped with an Intel Core i9-12900H processor, 64GB of RAM, and running Windows 11 Pro.
- **Device 2** The second device is a virtual machine hosted on a Dell OEMR XL R730 server, featuring 24 CPUs (Intel(R) Xeon(R) CPU E5-2680 v3, 2.50GHz). The virtual machine is configured with 16 CPU cores, 64 GB of memory, and also runs Windows 11 Pro.

The most significant hardware difference in this comparison lies in the CPU, specifically the maximum performance per core. Given that all implementations run exclusively on the CPU and not on the GPU, CPU performance is essential to overall training efficiency.

All algorithm versions prior to the PPO implementation rely heavily on NumPy, a CPU-bound computational library. Although the PPO implementation was built using Keras, which technically supports GPU acceleration, GPU usage was deliberately disabled to maintain comparability across all algorithms tested.

It is also important to consider the computational behaviour of the value-based algorithms, such as DDQL, where each training episode involves a large number of forward and backward passes through the NN. Due to the sequential nature of these operations, parallel execution through multithreading does not offer any performance advantage. In fact, the overhead of constantly switching between CPU cores would outweigh the benefits and actually slow down the computation. Consequently, all current implementations are designed to run single-threaded, utilising only one CPU core at a time.

In conclusion, due to the reliance on CPU-based libraries, the sequential execution of training operations, and the need for consistent comparability between models, single-core CPU performance plays a critical role in determining training speed and efficiency across different hardware setups.

Prototype	Episodes	Training Time D1 - Laptop	Training Time D2 - VM	Difference
v21 DDQL	100	1.7 min	3.6 min	+111.8%
v21 DDQL	1000	22.1 min	45.1 min	+104.1%
v21 DDQL	10'000	226.5 min	480.9 min	+112.3%
v23 SARSA	100	1.6 min	2.2 min	+37.5%
v23 SARSA	1000	22.1 min	23.9 min	+8.1%
v23 SARSA	10'000	277.4 min	670.5 min	+141.7%
v25 PPO	100	0.5 min	0.8 min	+60.0%
v25 PPO	1000	5.1 min	8.3 min	+62.7%
v25 PPO	10'000 (6000)	42.9 min	50.5 min	+17.7%

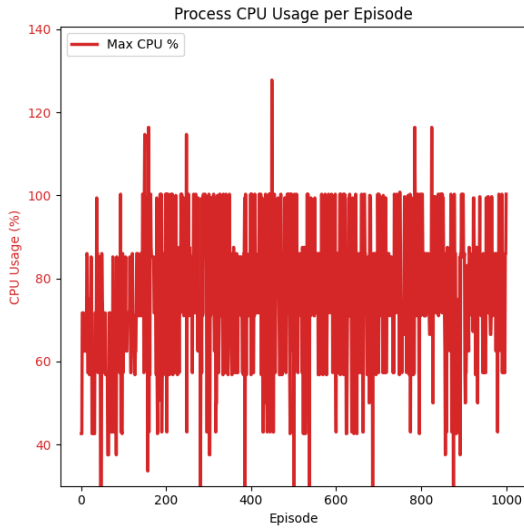
Table 6.3: Differences in Training Time between two Hardware setups

6.5.1 Performance Differences

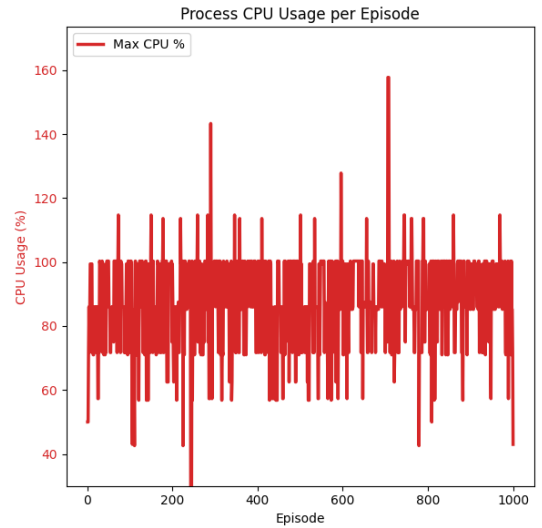
Table 6.3 compares the training times of various algorithm prototypes over different episode counts on two hardware setups. The results demonstrate that Device 1 (laptop) consistently outperforms Device 2 (virtual machine). On average, Device 1 trains approximately 109.4% faster for v21, 62.4% faster for v23, and 46.8% faster for v25, highlighting the significant impact of hardware architecture and CPU efficiency on training performance.

This observation is consistent with the previously discussed assertion that single-core CPU performance is the most critical factor in determining training speed. Since the algorithms rely primarily on CPU computation, and typically operate in a single-thread manner, due to the overhead associated with multithreading, the efficiency of each core becomes particularly important. While both device 1, the Dell XPS 9720 with an Intel Core i9-12900H, and device 2, the virtual machine with an Intel Xeon E5-2680 v3, have CPUs with a base frequency of 2.5 GHz, the Core i9 supports a maximum turbo boost of up to 5.0 GHz compared to the Xeon's peak of 3.3 GHz. This substantial difference in maximum clock speed, combined with the i9's newer architecture and improved instructions-per-cycle, contributes to the noticeably faster training performance observed on Device 1 40.

6.5.2 Consumption Differences



(a) CPU usage of D1 - Core i9-12900H



(b) CPU usage of D2 - Xeon E5-2680 v3

Figure 6.7: Comparison of CPU usage of PPO (v25) with 1,000 Episodes

An insightful observation from the CPU usage graph (Figure 6.7) is that the virtual machine (Device 2) shows slightly more consistent CPU usage across episodes compared to the main device (Device 1). At first glance, this might suggest that Device 2 provides more stable performance during training. However, this interpretation must be considered

in the context of the fundamental architectural differences between the two processors. This means that even when both machines are running at 100% CPU utilisation, their actual processing throughput differs significantly. In this context, 100% utilisation on Device 2 does not equate to the same processing power as 100% on Device 1. In addition, architectural improvements in the newer Intel i9 CPU, such as higher instruction-per-cycle throughput and more efficient core designs, further enhance its processing efficiency. These factors ultimately contribute to the superior training performance observed on Device 1, despite both systems displaying similar usage percentages.

Chapter 7

Final Considerations

7.1 Limitations

This thesis builds on the foundations laid by previous work, and as such, many of the previously identified limitations remain. Although certain areas have been explored in greater depth, numerous limitations could not be addressed due to the scope of this project and the necessity to maintain compatibility with previous experiments.

One major unresolved limitation is the lack of optimisation of online training speed and data handling. These aspects depend on the original client-side infrastructure, which was neither redesigned nor extended in this study. In addition, no new behavioural data were collected, meaning that the RL agents continued to train on the same static dataset. This reliance on fixed data further restricts the system’s capacity for dynamic learning.

The AD model employed in this project was also left unchanged to ensure fair comparability with the previous results. Although this approach benefits reproducibility, it also leaves the limitations of the AD system, including potential weaknesses in its detection sensitivity or response behaviour, unaddressed.

In addition to these inherited problems, several new challenges emerged during the course of the thesis. In particular, substantial computational constraints were observed. As discussed in Chapter [6.5.1](#), all RL agents were implemented using NumPy, which performs computations on a single CPU thread. In contrast to frameworks such as PyTorch, which support parallelism and hardware acceleration, NumPy lacks native capabilities for exploiting multicore processors or GPUs. Consequently, the frequent and sequential value updates required in each episode led to inefficient CPU usage and high processing overhead. This inefficiency was particularly evident in DDQL, which required intensive computational operations per episode. A potential improvement would involve adopting batch processing techniques (e.g., grouped updates or ”baskets”) to reduce redundant operations and better exploit available processing resources.

For algorithms such as PPO, performance could be significantly improved by migrating to a GPU-accelerated environment. Parallel processing would allow multiple computations

to be performed simultaneously, thus reducing training time and improving throughput, especially in environments with high-dimensional state spaces.

Another important limitation lies in the narrow scope of the evaluation environment. This work focused exclusively on ransomware targeting spectrum sensors as an IoT device type. However, the IoT landscape encompasses a wide range of devices, such as smart home appliances, wearables, or industrial sensors, each with different behavioural patterns and security architectures. Similarly, other types of malware, such as spyware, rootkits, or denial-of-service agents, operate differently and may require different learning strategies. Consequently, the effectiveness of the RL algorithms explored in this work for broader cyber security threats remains untested.

In conclusion, while this work makes a valuable contribution by comparing multiple RL algorithms in a security context, its applicability is limited by computational constraints, reliance on static data, architectural choices, and a narrow focus on specific devices and malware types. Addressing these limitations in future work will be essential for the development of more robust and generalisable AI-driven cyber security tools.

7.2 Summary and Conclusions

This thesis builds on the RansomAI framework introduced in [8], which demonstrated a PoC ransomware trained using the DQL algorithm. The main objective of this work was to explore whether alternative RL algorithms could further optimise ransomware behaviour, enhance learning efficiency, and prove more suitable for this uniquely adversarial environment.

The initial experimental setup consisted of an IoT device equipped with spectrum sensors, acting as the ransomware client, alongside an AD system for monitoring. Communication with the C&C server was facilitated through behavioural fingerprints extracted from the IoT device and transmitted via an API. To ensure comparability with the original framework and limit the scope of the implementation, components such as the AD system and the fingerprinting mechanism were retained without modification from the original setup.

In this work, three new RL algorithms were implemented and systematically evaluated: DDQL, SARSA, and PPO. Their performance was assessed using a range of metrics, including accuracy, training speed, sample efficiency, resource consumption, convergence behaviour, and overall reward.

Firstly, DDQL was introduced in versions 20 and 21 as an extension of the existing DQL approach. Its primary innovation was the inclusion of a separate target network to increase overall training stability and mitigate the feedback loops that frequently lead to instability in traditional Q-Learning. This structural improvement allowed DDQL to better estimate action values, thereby reducing the overestimation bias commonly observed in DQL implementations.

Empirical results demonstrated that DDQL outperformed its predecessor in several fronts. In addition to maintaining a high level of accuracy, DDQL showed faster convergence,

improved sample efficiency, and a more consistent training trajectory. These advantages translated into smoother learning curves and reduced variance in performance across training runs. Although the overall accuracy gains were relatively small, due to the already strong performance of DQL, the increased stability made the training process more predictable and reliable.

One practical outcome of these enhancements was a notable reduction in training time. On average, DDQL completed the training process of 1,000 episodes in 22.1 minutes, compared to 42.2 minutes for DQL. This reduction highlights the efficiency gains achieved through improved bias management and algorithmic stability. These findings suggest that DDQL offers a more robust solution for training agents in complex environments where consistency and computational efficiency are critical.

In contrast, the implementation of the tabular SARSA algorithm served as a baseline for evaluating the capabilities and limitations of traditional, non-neural RL techniques. However, SARSA failed to identify viable ransomware configurations or produce meaningful training outcomes, demonstrating the limitations of tabular methods in complex, high-dimensional environments. Additionally, these findings reaffirm the necessity of neural-based architectures for addressing non-linear, real-world problem spaces.

To further explore the potential of alternative RL strategies, PPO, a policy gradient-based algorithm, was implemented to test whether policy-based methods could outperform value-based methods. The results confirmed this hypothesis: PPO consistently achieved high accuracy in significantly fewer training episodes, demonstrating high sample efficiency and stable performance over a wide range of training runs. Its use of direct policy updates enabled PPO to reach 100% accuracy in less than 50 episodes, whereas DDQL, although more computationally efficient over the same time period, required over 200 episodes to surpass 90% accuracy.

Lastly, an additional investigation was conducted to assess the impact of hardware performance on training efficiency. A comparative analysis between two different hardware setups revealed a notable performance gap of approximately 70%, a significant finding with implications for long-term or large-scale training scenarios.

7.3 Future Work

As outlined in Section [7.1](#), several areas could be improved and extended in future work to address the identified limitations and further enhance the capabilities of the proposed approach. One particular persistent challenge is the improvement of the AD capabilities.

In the context of adversarial RL, applying AD presents an inherent conflict in design objectives. On the one hand, a high-quality AD system is critical for creating an effective "opponent" to train against, thereby enabling the agent to develop sophisticated evasion strategies. On the other hand, from the attacker's perspective, weaker AD performance is preferable, as it facilitates more consistent ransomware success, reduces fluctuations between detected and hidden behaviours, and thus simplifies the agent's learning process.

This trade-off presents an interesting challenge: while stronger AD performance is desirable to improve detection accuracy and evasion resistance, it may simultaneously limit the flexibility in configuring ransomware behaviours. Future research should focus on optimising this trade-off, ideally tending toward a decision in favour of improved AD performance. Doing so would not only elevate detection accuracy but also positively impact the overall effectiveness of the cyber defence.

Furthermore, allowing the agent to dynamically construct and select optimal ransomware configurations, rather than relying on a fixed predefined set, could significantly enhance its learning process and better reflect real-world conditions. By allowing the agent to autonomously adjust key parameters, such as burst duration, burst pause, and encryption speed, within defined limits, the system would support more nuanced and adaptive behaviour. Additionally, this flexibility would increase the robustness and realism of both the offensive and defensive mechanisms.

Another promising direction for future research would be to expand the range of RL algorithms investigated, including hybrid methods such as SAC, that combine the strengths of different learning paradigms. With a foundational comparison of current RL algorithms established in this work, further investigation into more advanced or composite RL strategies could provide valuable insights into their adaptability and performance in more complex or varied threat scenarios.

Equally important is the need to broaden the current simulation environment. The current research has been limited to the evaluation of ransomware-targeting spectrum sensors, which represent a narrow subset of IoT devices. Future investigations should expand the testbed to include additional device types, such as smart home appliances, wearable devices, or industrial control systems, to enable a more comprehensive evaluation of the approach. In addition, introducing diverse types of malware, such as spyware, rootkits, or denial-of-service agents, would provide a more thorough assessment of whether the learning strategies developed in this study remain effective across a wider spectrum of cyber threats.

Pursuing the aforementioned lines of inquiry would not only improve the external validity of the proposed approach, but also provide valuable insights into the adaptability of RL-based cyber agents to diverse behavioural patterns and security architectures, thus improving their generalisation capability in real-world cyber security applications.

Bibliography

- [1] L. S. Vailshery, *Number of Internet of Things (IoT) connected devices worldwide from 2022 to 2023, with forecasts from 2024 to 2033*, Last Visit 20.01.2025, 2024. [Online]. Available: <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/>.
- [2] M. M. Hossain, M. Fotouhi, and R. Hasan, "Towards an Analysis of Security Issues, Challenges, and Open Problems in the Internet of Things". In: *2015 IEEE World Congress on Services*. 2015, pp. 21–28. DOI: [10.1109/SERVICES.2015.12](https://doi.org/10.1109/SERVICES.2015.12). [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7196499/citations#citations>.
- [3] R. Ahmad and I. Alsmadi, "Machine learning approaches to IoT security: A systematic literature review". In: *Internet of Things 14* (2021), p. 100365. ISSN: 2542-6605. DOI: <https://doi.org/10.1016/j.iot.2021.100365>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2542660521000093>.
- [4] R. Daws, *UK introduces first IoT security laws*, Last Visit 20.01.2025, 2024. [Online]. Available: <https://iotttechnews.com/news/uk-introduces-first-iot-security-laws/>.
- [5] L. Fritsch, A. Jaber, and A. Yazidi, "An Overview of Artificial Intelligence Used in Malware". In: *Nordic Artificial Intelligence Research and Development*. Ed. by E. Zouganeli, A. Yazidi, G. Mello, et al. 2022, pp. 41–51. ISBN: 978-3-031-17030-0.
- [6] J. von der Assen, A. H. Celdrán, J. Luechinger, et al., *RansomAI: AI-powered Ransomware for Stealthy Encryption*, 2023. arXiv: [2306.15559](https://arxiv.org/abs/2306.15559) [cs.CR]. [Online]. Available: <https://arxiv.org/abs/2306.15559>.
- [7] S. Padovan, "AI-powered Ransomware to Stay Hidden". Last Visit 21.01.2025. Master's Thesis. University of Zurich, 2024. [Online]. Available: <https://files.ifi.uzh.ch/CSG/staff/vonderassen/extern/theses/ma-padovan.pdf>.
- [8] J. Luechinger, "AI-powered Ransomware to Optimize its Impact on IoT Spectrum Sensors". MA thesis. University of Zurich, Mar. 2023. [Online]. Available: <https://doi.org/10.5167/uzh-255754>.

- [9] N. Kaloudi and J. Li, “The AI-Based Cyber Threat Landscape: A Survey”. In: *ACM Comput. Surv.* 53.1 (Feb. 2020). ISSN: 0360-0300. DOI: [10.1145/3372823](https://doi.org/10.1145/3372823). [Online]. Available: <https://doi.org/10.1145/3372823>.
- [10] J. Ferdous, R. Islam, A. Mahboubi, et al., “AI-Based Ransomware Detection: A Comprehensive Review”. In: *IEEE Access* 12 (2024), pp. 136666–136695. DOI: [10.1109/ACCESS.2024.3461965](https://doi.org/10.1109/ACCESS.2024.3461965).
- [11] S. Rajendran, R. Calvo-Palomino, M. Fuchs, et al., “Electrosense: Open and Big Spectrum Data”. In: *IEEE Communications Magazine* 56.1 (Jan. 2018), 210–217. ISSN: 0163-6804. DOI: [10.1109/mcom.2017.1700200](https://doi.org/10.1109/mcom.2017.1700200). [Online]. Available: <http://dx.doi.org/10.1109/MCOM.2017.1700200>.
- [12] P. M. S. Sánchez, A. H. Celdrán, G. Bovet, et al., “SpecForce: A Framework to Secure IoT Spectrum Sensors in the Internet of Battlefield Things”. In: *IEEE Communications Magazine* 61.5 (2023), pp. 174–180. DOI: [10.1109/MCOM.001.2200349](https://doi.org/10.1109/MCOM.001.2200349).
- [13] P. M. S. Sanchez, J. M. J. Valero, A. H. Celdran, et al., “A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets”. In: *IEEE Communications Surveys & Tutorials* 23.2 (2021), 1048–1077. ISSN: 2373-745X. DOI: [10.1109/comst.2021.3064259](https://doi.org/10.1109/comst.2021.3064259). [Online]. Available: <http://dx.doi.org/10.1109/COMST.2021.3064259>.
- [14] B. A. S. Al-rimy, M. A. Maarof, and S. Z. M. Shaid, “Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions”. In: *Computers & Security* 74 (2018), pp. 144–166. ISSN: 0167-4048. DOI: <https://doi.org/10.1016/j.cose.2018.01.001>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S016740481830004X>.
- [15] Kaspersky, *Every third cyber incident was due to ransomware, Kaspersky reports*, Last Visit 29.01.2025, 2023. [Online]. Available: <https://www.kaspersky.com/about/press-releases/every-third-cyber-incident-was-due-to-ransomware-kaspersky-reports>.
- [16] H. Oz, A. Aris, A. Levi, et al., “A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions”. In: *ACM Comput. Surv.* 54.11s (Sept. 2022). ISSN: 0360-0300. DOI: [10.1145/3514229](https://doi.org/10.1145/3514229). [Online]. Available: <https://doi.org/10.1145/3514229>.
- [17] P. H. Meland, Y. F. F. Bayoumy, and G. Sindre, “The Ransomware-as-a-Service economy within the darknet”. In: *Computers & Security* 92 (2020), p. 101762. ISSN: 0167-4048. DOI: <https://doi.org/10.1016/j.cose.2020.101762>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167404820300468>.
- [18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Second. 2018. [Online]. Available: <http://incompleteideas.net/book/the-book-2nd.html>.

- [19] OpenAI, *Introduction to Reinforcement Learning - Spinning Up*. Accessed: 2025-01-25. 2025. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html.
- [20] H. Wang, T. Zariphopoulou, and X. Zhou, *Exploration versus exploitation in reinforcement learning: a stochastic control approach*, 2019. arXiv: [1812.01552](https://arxiv.org/abs/1812.01552) [math.OC]. [Online]. Available: <https://arxiv.org/abs/1812.01552>.
- [21] A. dos Santos Mignon and R. L. de Azevedo da Rocha, "An Adaptive Implementation of ϵ -Greedy in Reinforcement Learning". In: *Procedia Computer Science* 109 (2017), pp. 1146–1151. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2017.05.431>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050917311134>.
- [22] H. Dong, Z. Ding, and S. Zhang, eds., *Deep Reinforcement Learning: Fundamentals, Research and Applications*. 2020. ISBN: 978-981-15-4094-3. DOI: [10.1007/978-981-15-4095-0](https://link.springer.com/book/10.1007/978-981-15-4095-0). [Online]. Available: <https://link.springer.com/book/10.1007/978-981-15-4095-0>.
- [23] GeeksforGeeks, *Types of Reinforcement Learning*. Accessed: 2025-01-20. 2024. [Online]. Available: <https://www.geeksforgeeks.org/types-of-reinforcement-learning/>.
- [24] V. Mnih, K. Kavukcuoglu, D. Silver, et al., *Playing Atari with Deep Reinforcement Learning*, 2013. arXiv: [1312.5602](https://arxiv.org/abs/1312.5602) [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1312.5602>.
- [25] W Schultz, P Dayan, and P. Montague, "Responses of monkey dopamine neurons to reward and conditioned stimuli during successive steps of learning a delayed response task." In: *Science* 275.5306 (1993), pp. 1593–1599. DOI: [10.1126/science.275.5306.1593](https://pubmed.ncbi.nlm.nih.gov/8441015/). [Online]. Available: <https://pubmed.ncbi.nlm.nih.gov/8441015/>.
- [26] S. Hassan, "STUDY OF ARTIFICIAL INTELLIGENCE IN CYBER SECURITY AND THE EMERGING THREAT OF AI-DRIVEN CYBER ATTACKS AND CHALLENGE". In: *SSRN Electronic Journal* (Jan. 2023). DOI: [10.2139/ssrn.4652028](https://ssrn.com/abstract=4652028).
- [27] M. Alanezi and R. M. A. AL-Azzawi, "AI-Powered Cyber Threats: A Systematic Review". In: *Mesopotamian Journal of CyberSecurity* 4.3 (Dec. 2024), 166–188. DOI: [10.58496/MJCS/2024/021](https://journals.mesopotamian.press/index.php/CyberSecurity/article/view/648). [Online]. Available: <https://journals.mesopotamian.press/index.php/CyberSecurity/article/view/648>.
- [28] S. Okdem and S. Okdem, "Artificial Intelligence in Cybersecurity: A Review and a Case Study". In: *Applied Sciences* 14.22 (2024). ISSN: 2076-3417. DOI: [10.3390/app142210487](https://www.mdpi.com/2076-3417/14/22/10487). [Online]. Available: <https://www.mdpi.com/2076-3417/14/22/10487>.

- [29] CyberPeace, *Research Report: AI-Powered Ransomware Attack on a Healthcare Provider*, Last Visit 01.02.2025, 2024. [Online]. Available: <https://www.cyberpeace.org/resources/blogs/research-report-ai-powered-ransomware-attack-on-a-healthcare-provider>.
- [30] H. S. Anderson, A. Kharkar, B. Filar, et al., *Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning*, 2018. arXiv: 1801.08917 [cs.CR]. [Online]. Available: <https://arxiv.org/abs/1801.08917>.
- [31] D. Commey, B. Appiah, B. K. Frimpong, et al., “EGAN: Evolutional GAN for Ransomware Evasion”. In: *2023 IEEE 48th Conference on Local Computer Networks (LCN)*. Oct. 2023, 1–9. DOI: 10.1109/lcn58197.2023.10223320. [Online]. Available: <http://dx.doi.org/10.1109/LCN58197.2023.10223320>.
- [32] T. Jiang, Y. Liu, X. Wu, et al., “Application of deep reinforcement learning in attacking and protecting structural features-based malicious PDF detector”. In: *Future Generation Computer Systems* 141 (2023), pp. 325–338. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2022.11.015>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X22003740>.
- [33] T. Quertier, B. Marais, S. Morucci, et al., *MERLIN – Malware Evasion with Reinforcement LearnINg*, 2022. arXiv: 2203.12980 [cs.CR]. [Online]. Available: <https://arxiv.org/abs/2203.12980>.
- [34] T. Meyer, N. Kaloudi, and J. Li, “A Systematic Literature Review on Malicious Use of Reinforcement Learning”. In: *2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*. 2021, pp. 21–28. DOI: 10.1109/EnCyCriS52570.2021.00011.
- [35] L. Gao, “Comparison of DQN And Double DQN Reinforcement Learning Algorithms for Stock Market Prediction”. In: *Proceedings of the 2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)*. 2024, pp. 169–177. ISBN: 978-94-6463-370-2. DOI: 10.2991/978-94-6463-370-2_19. [Online]. Available: https://doi.org/10.2991/978-94-6463-370-2_19.
- [36] M.-A. Chadi and H. Mousannif, *Understanding Reinforcement Learning Algorithms: The Progress from Basic Q-learning to Proximal Policy Optimization*, 2023. arXiv: 2304.00026 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2304.00026>.
- [37] H. van Hasselt, A. Guez, and D. Silver, *Deep Reinforcement Learning with Double Q-learning*, 2015. arXiv: 1509.06461 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1509.06461>.
- [38] Z. Wang, T. Schaul, M. Hessel, et al., *Dueling Network Architectures for Deep Reinforcement Learning*, 2016. arXiv: 1511.06581 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1511.06581>.

- [39] T. Schaul, J. Quan, I. Antonoglou, et al., *Prioritized Experience Replay*, 2016. arXiv: [1511.05952](https://arxiv.org/abs/1511.05952) [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1511.05952>.
- [40] CPU-Monkey, *Intel Xeon E5-2680 v3 vs Intel Core i9-12900H – Benchmark and Specs Comparison*, Last Visit 09.05.2025, 2024. [Online]. Available: https://www.cpu-monkey.com/en/compare_cpu-intel_xeon_e5_2680_v3-vs-intel_core_i9_12900h.

Abbreviations

AD	Anomaly Detection
AES	Advanced Encryption Standard
AI	Artificial Intelligence
API	Application Programming Interface
C&C	Command and Control
CPU	Central Processing Unit
CSG	Communication Systems Group
DDQL	Double Deep Q-Learning
DL	Deep Learning
DNN	Deep Neural Network
DoS	Denial of Service
DP	Dynamic Programming
DQL	Deep Q-Learning
DRL	Deep Reinforcement Learning
EGAN	Evolutionary Generative Adversarial Network
ES	Evolution Strategy
FDI	False Data Injection
FP	Fingerprint
GA	Genetic Algorithm
GAE	Generalized Advantage Estimation
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
IoT	Internet of Things
MC	Monte Carlo
MDP	Markov Decision Process
ML	Machine Learning
NN	Neural Network
PDF	Portable Document Format
PE	Preinstallation Environment
PER	Prioritised Experience Replay
PoC	Proof of Concept
PPO	Proximal Policy Optimization
RaaS	Ransomware-as-a-Service
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RF	REINFORCE

RL	Reinforcement Learning
RSA	Rivest–Shamir–Adleman
SAC	Soft Actor-Critic
SARSA	State-Action-Reward-State-Action
SDR	Software-Defined Radio
SGD	Stochastic Gradient Descent
SiLU	Sigmoid Linear Unit
TD	Temporal Difference
TLS	Transport Layer Security
VM	Virtual Machine

List of Figures

2.1 Overview of the RL structure [18]	8
2.2 Response of dopamine neurons [18]	16
3.1 Heat map for the impact of AI-powered cyber threats across different sectors [27]	19
4.1 Overview of the Client - Server Environment [8]	35
5.1 Overview of the reward functions	39
5.2 Overview of the different activation functions	40
5.3 Performance Results of v21 after 10,000 Episodes	44
5.4 Architecture of the neural network for the PPO agent	51
5.5 Performance Results of v25 after 2,000 Episodes	53
6.1 Average time needed per episode for v20 and v21	56
6.2 CPU usage comparison of DDQL algorithms over 1,000 Episodes	57
6.3 Performance analysis of DDQL algorithms over 1,000 Episodes	58
6.4 Memory usage over 10,000 Episodes of SARSA v23	59
6.5 Comparison of CPU usage of DDQL and PPO algorithms	62
6.6 Comparison of Memory usage of DDQL and PPO algorithms	63
6.7 Comparison of CPU usage of PPO (v25) with 1,000 Episodes	65

List of Tables

3.1	Review and Comparison of Related Literature	23
4.1	Comparison of Basic RL Algorithms	28
4.2	Review and Comparison of DRL Algorithms [18, 22, 35-38]	32
4.3	Ransomware Configurations [6]	36
5.1	DDQL – Overview of Prototype Differences	42
5.2	DDQL - Training performance across different episode lengths	43
5.3	Accuracy results for Prototype 21 after 10,000 episodes $\epsilon = 0.4$, $\delta = 0.01$, $\alpha = 0.005$, $\gamma = 0.9$	45
5.4	SARSA – Overview of Prototype Differences	45
5.5	Accuracy results for Prototype 23 after 10,000 episodes $\epsilon = 0.4$, $\delta = 0.1$, $\alpha = 0.05$, $\gamma = 0.9$	47
5.6	PPO – Overview of Prototype Differences	48
5.7	PPO - Training performance across different episode lengths	53
6.1	DQL and DDQL - Training performance across different episode lengths	58
6.2	Performance results comparison of DDQL and PPO	61
6.3	Differences in Training Time between two Hardware setups	64

List of Equations

2.1 Deterministic Policy Notation	8
2.2 Stochastic Policy Notation	8
2.3 Cumulative Discounted Reward Equation	10
2.4 Basic "every-visit" Monte Carlo Equation	11
2.5 simple TD method called TD(0) or one-step TD	11
2.6 Definition of Q-Learning Algorithm	13
2.7 Update rule of REINFORCE	14
4.1 SARSA Update Rule	26
4.2 DDQL - Action Selection	29
4.3 DDQL - Target Q-Value Calculation	29
4.4 Definition of Q-Learning Algorithm	30
5.1 Reward when remains Hidden - Old	38
5.2 Reward when detected - Old	38
5.3 Reward when remains Hidden - Current	39
5.4 Reward when detected - Current	39
5.5 Min-Max-Scaling	41

List of Algorithms

5.1 Double Deep Q-Learning (DDQL)	43
5.2 Pseudocode for Tabular SARSA	46
5.3 Pseudocode for PPO Agent	48

Listings

<code>5.1 PPO Neural Network Model</code>	49
<code>5.2 PPO action selection method</code>	52

Appendix A

Contents of the Repository

The Repository with the implementation for the C&C Server is available under the following URL:

- C&C Server https://github.com/lguebeli/novel_roar_server

To use the implementation, please refer to the guidelines provided in the **README** and **INSTALL** files.