university of groningen

faculty of science and engineering

Multi-Agent Reinforcement Learning for Cyber Defence

Bachelor's Project Computing Science

July 2025

Author: Manos Savvides
Student Number: s5106389
First supervisor: Dr. Fatih Turkmen
Second supervisor: R. Fernandes Cunha

Abstract

In the last decade, the increase of cyberattacks has precipitated substantial destruction within the technological sector. Consequently, there has been an imperative for the development of advanced defence mechanisms that are capable of detecting and averting such attacks from inflicting harm. Recent studies have increasingly focused on Reinforcement Learning as a means to mitigate the impact of these cyberattacks. These studies involve conducting diverse experiments in various simulated environments and evaluating their results. Despite the notable advances achieved, significant challenges persist in identifying the most effective models, policies, and methodologies for each distinct scenario. The present study will employ and compare collaborative reinforcement learning, specifically Proximal Policy Optimisation (PPO) based MARL algorithms within the complex, simulated CybORG environment, with the aim of identifying optimal defence mechanisms. The research is expected to contribute towards improved accuracy and adaptability of defence responses against complex cyber threats. We show that MAPPO outperforms IPPO through the use of hyperparameter tuning. Although the hierarchical MARL (HMARL) agent learns faster in the initial phase of the simulation, its final performance is worse than that of the tuned non-hierarchical agents. These findings indicate improved accuracy and adaptability of defence responses to complex cyber threats.

Contents

List of Figures

List of Tables

1 Introduction

Recent years have seen a marked increase in the number of cyberattacks, with profound consequences for the global economy. In 2020, the cost to the global economy was 1 trillion USD, representing an increase of more than 50% since 2018 [10]. Cyberattacks are increasingly leveraging Artificial Intelligence (AI) driven automation and adaptive strategies. Conventional methods are not always adequate in countering such attacks because AI attacks are rapid and versatile. This creates a gap between an organisation's security mechanisms, which are usually reactive and not being developed at the same pace as that of the attacker [9]. Recent comparative analyses of traditional and AI-driven approaches have highlighted the considerable potential of AI-based defence mechanisms [21, 33].

The necessity for novel cyberdefence techniques is as pressing as it has ever been, with a plethora of studies emerging that focus on Machine Learning (ML) as a means of combating cybercrime [18]. Considerable attention has been paid to the application of Reinforcement Learning (RL) in the context of automating cybersecurity defence mechanisms [16]. Among the various ML techniques, RL is distinguished by its capacity to model adversarial interactions as sequential decision-making problems, rendering it particularly well-suited for cyber defence applications. Indeed, the sequential decision-making framework of RL uniquely addresses the limitations of traditional methods by enabling systems to continually learn and adapt in real time, proactively optimising defence strategies as new threats emerge. The findings indicate that RL algorithms have the capacity to learn optimal defence strategies and play a crucial role in enhancing the security of a system [31, 20].

It is imperative to analyse a range of techniques and practices in complex environments where multiple agents collaborate to defend a system and also to investigate the potential utilisation of these techniques in real-world scenarios. The focus of this study is to identify the most effective methods for achieving optimal results in mitigating damages from network attacks. In pursuit of answering this question, we will analyse various approaches within a simulated scenario specifically designed to replicate network attack conditions. In particular, our study is conducted within the context of the CAGE Challenge 4 [32]. The overarching goal is to build upon existing strategies capable of effectively reducing or preventing potential damages. It is important to acknowledge that the simulation phase presented here constitutes an essential preliminary step, as deploying these RL mechanisms directly into a real network scenario would otherwise be impractical without prior validation and assessment through simulation. The two main research questions of this study are:

- **Research Question 1:** Identify the optimal hyperparameters for each Proximal Policy Optimisation (PPO) algorithm and subsequently evaluate their comparative performance to determine the most effective approach.

- **Research Question 2:** Investigate the potential advantages of Hierarchical Multi-Agent Reinforcement Learning (HMARL) algorithms in cyber defence contexts.

2 Background

In this section, we draw on concepts from Reinforcement Learning: An Introduction by Sutton and Barto [30] and Multi-Agent Reinforcement Learning: Foundations and Modern Approaches by Albrecht et al. [3] to help readers understand our methods. In addition, we will analyse the current state of the art and reference existing implementations in Multi-Agent reinforcement learning (MARL) scenarios. Finally, we present a detailed overview of the environment we work on.

2.1 Reinforcement Learning

RL is a subset of ML that involves learning through interaction with the environment, as illustrated in Figure 1. Feedback from the environment influences subsequent decisions, which is a fundamental aspect of intelligence. Examples include learning to drive or ride a bicycle. The objective is to learn the optimal policy, which is a mapping from states to actions in order to maximise the expected cumulative reward (return). There are two main elements: the agent and the environment. The environment is the space that the agent lives and has to interact with. At each step t, the agent receives observations from the environment S_t and selects an action A_t. Based on the selected action, the environment transitions to a new state S_{t+1} resulting in a reward of r_{t+1}.

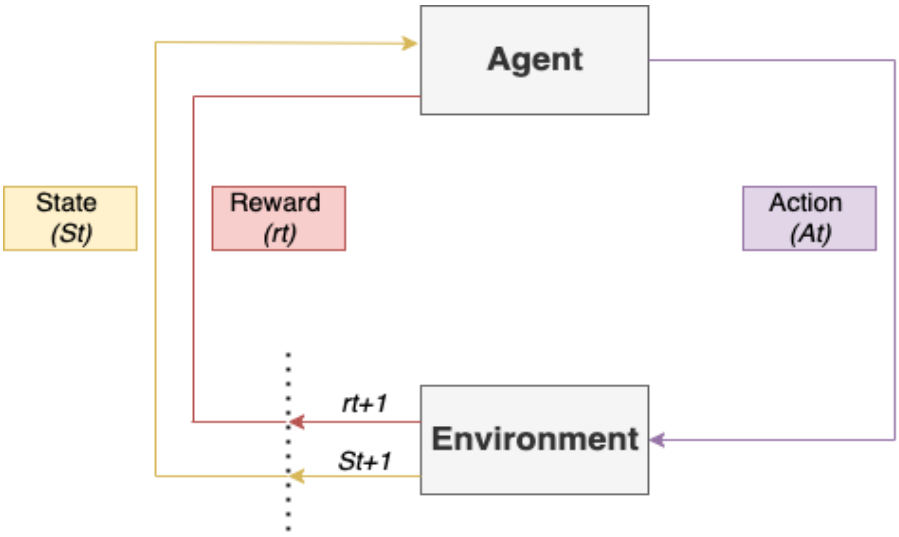


Figure 1: Reinforcement learning agent-environment interaction.

2.1.1 States and Observations

A state at timestep t, denoted by S_t, represents a description of the world at a specific point in time. In more complex environments, the agent may not have access to all the information about the state; in such cases, this is represented by an observation O_t. When the agent has access to all information about the environment, the environment is said to

be **fully observable**. Conversely, when the agent can only perceive partial information, the environment is considered **partially observable**.

2.1.2 Action spaces

There are two types of action spaces in reinforcement learning: discrete and continuous. A discrete action space consists of a finite number of valid actions, as seen in games like chess. In contrast, a continuous action space involves an infinite number of possible actions, such as those required for controlling a robotic arm. This has a significant impact on the choice of algorithm used to solve a problem, as some algorithms are not compatible with both types of action spaces.

2.1.3 Policies

A policy is a set of rules that determines which action an agent should take in a given state to achieve a specific goal. This is a fundamental concept, as the agent interacts with the environment in order to learn the optimal policy. A policy can be deterministic, typically denoted as $A_t = \mu(S_t)$, where μ is the policy function and the output is the same action as shown in Figure 2. Alternatively, the policy can be stochastic, where the output is a probability distribution over actions. This is usually expressed as $A_t \sim \pi(\cdot \mid S_t)$.

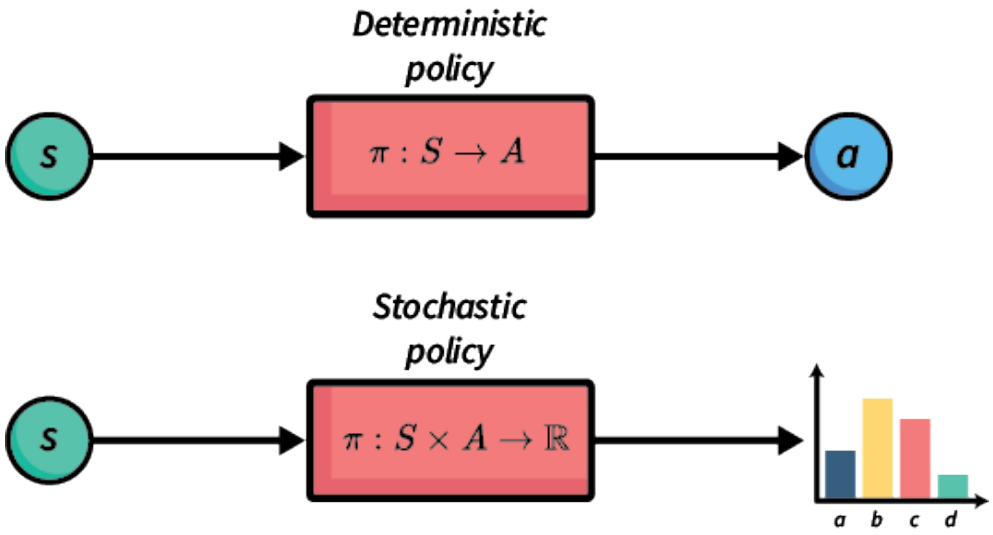


Figure 2: Difference between deterministic and stochastic policy. [5]

2.1.4 Trajectories

A trajectory, denoted by $\tau = (s_0, a_0, s_1, a_1, \dots)$, is a sequence of states and actions in the environment. Trajectories are often referred to as episodes, and this terminology will be used in the following sections.

2.1.5 Rewards

The reward function r plays a crucial role in reinforcement learning. In this study, we define $r(s, a, s')$ as the reward received when transitioning from state s to state s' after taking action a. At time step t, this is written as $r_t = r(s_t, a_t, s_{t+1})$. The objective of the agent is to maximise the cumulative reward over an episode, known as the *return*. The return can be defined as:

- **Undiscounted:** $G(\tau) = \sum_{t=0}^{T} r_t$, where $r_t = r(s_t, a_t, s_{t+1})$ is the reward at time step t.
- **Discounted:** $G(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in (0, 1)$ is the discount factor and $r_t = r(s_t, a_t, s_{t+1})$.

The discounted return reflects the intuition that immediate rewards are typically preferred over future rewards.

2.1.6 Markov Decision Process (MDP)

A Markov Decision Process (MDP) is a fundamental framework in reinforcement learning for modelling sequential decision-making problems. Each state satisfies the *Markov Property*, meaning the agent only requires knowledge of the current state to make decisions. A discounted MDP is formally defined as a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where:

- $\mathcal{S}$: the state space.
- $\mathcal{A}(s) \subseteq \mathcal{A}$: the set of actions available in each state $s \in \mathcal{S}$.
- $P(s' \mid s, a)$: the transition probability to state s' when taking action a in state s.
- $r(s, a, s')$: the reward received when transitioning from state s to state s' using action a.
- $\gamma \in [0, 1)$: the discount factor.

An episode begins in an initial state S_0 sampled from an initial state distribution $p(S_0)$ over $\mathcal{S}$.

The MDP is a fully observable, probabilistic state model. However, when dealing with partially observable environments, a variation called a Partially Observable Markov Decision Process (POMDP) is used [28]. A POMDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, P, r, O, \Omega, \gamma)$, which extends the standard MDP by including:

- O: an observation function; the probability of receiving observation o, when transitioning from state s using action a.
- Ω: is a finite set of observations.

Since the true state is hidden in a POMDP, these changes force the agent to make decisions based on observations, as it no longer has full knowledge of the state. An episode begins in the initial belief state b_0, a probability distribution over the state space S that represents the uncertainty about the true starting state.

2.1.7 Deep Reinforcement Learning (DRL)

Deep Reinforcement Learning (DRL) is a type of traditional reinforcement learning (RL) that employs neural networks (NNs). Compared to supervised learning, these networks typically have fewer parameters and layers, due to the added complexity inherent in RL environments [12]. Neural networks in DRL can be used to represent value functions, policies, or models of the environment. As shown in Figure 3, a neural network in an RL setting receives a state as input and outputs a probability distribution over possible actions. In this context, the NN represents the policy.

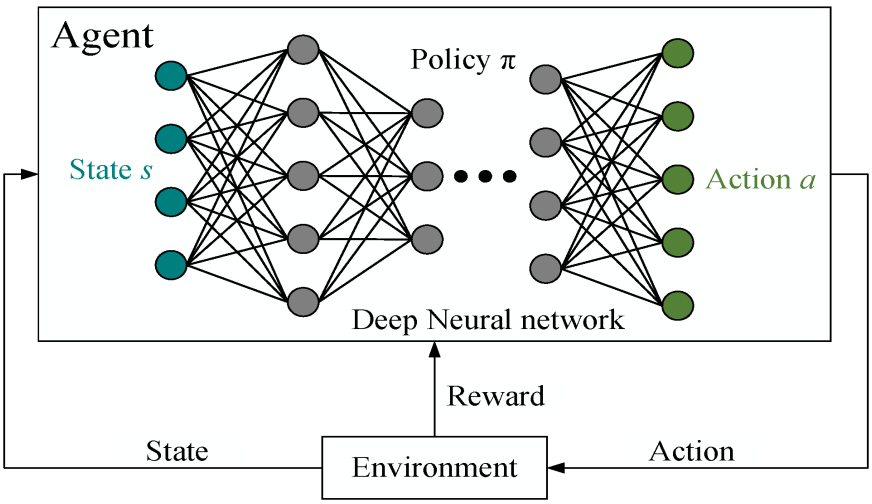


Figure 3: Basic Principle of DRL [38].

2.2 Multi-Agent Reinforcement Learning

In this section, we will briefly explain an extension of regular RL, known as Multi-Agent reinforcement learning (MARL). The main difference is that there are now multiple agents that must be trained, making the training process more complex. In real-world scenarios, a single agent cannot handle all decisions and actions; MARL addresses this issue by allowing each agent to manage a specific subproblem. Adding or removing agents increases flexibility, as each agent can learn its own policy. In certain environments, agents have the possibility to share information, which may benefit learning and lead to increased stability. Since we are investigating network attacks, we will utilise MARL, assigning each agent responsibility for a specific part of the network. There are three main paradigms in interactions between the agents. In a **cooperative** configuration, agents work together towards a common goal and

aim to find the optimal policy for the entire system. On the other hand, in a **competitive** configuration, agents compete with each other aiming only to maximise their own individual rewards and minimize the rewards of other agents. This scenario is typical in zero-sum games like chess or poker, where one agent's gain directly equals another agent's loss. Finally the third paradigm called **mixed** configuration. In this paradigm, agents may form alliances but can also compete against each other. This is used in scenarios where some agents share the same goals, while others have conflicting objectives. Figure 4 illustrates an example of such a setting: a simplified network attack scenario where red agents attempt to attack two servers, while blue agents defend them.

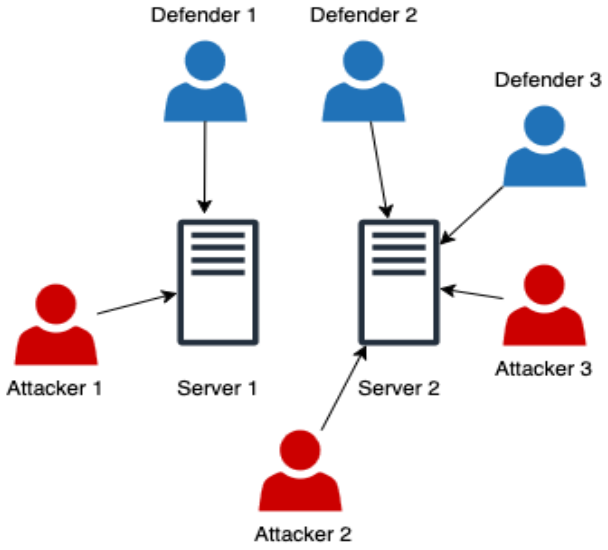


Figure 4: Mixed setting in multi-agent reinforcement learning for the defence and attack of a server.

Since we now have many agents, there are multiple ways to train them. We will briefly explain the methods used in this research. The simplest approach is **Independent Learning**, where each agent learns its own policy without considering the observations or actions of other agents. Alternatively, in **Centralised Training with Decentralised Execution (CTDE)**, a centralised critic leverages global state information during training, while each agent (actor) relies only on its local observations during execution. Finally, in **Hierarchical approaches**, agents operate using a multi-level policy structure. Typically, a high-level policy selects abstract actions, while a low-level policy executes the corresponding primitive actions. This is illustrated in Figure 5, where the master policy receives an observation and then selects a sub-policy, which in turn chooses the appropriate primitive action.

We will also outline the most common challenges associated with MARL. **Non-stationarity** makes it difficult for an agent to converge to an optimal policy, as the policies of other agents are constantly changing. **Scalability** becomes a concern as the number of agents increases, causing the state and action spaces to grow significantly, which makes training computationally expensive. Designing communication protocols for **communication and coordination** is also a major challenge, as there are many possible approaches, and it is not always clear which method is most effective. Finally, when using a global reward scheme, it can be hard

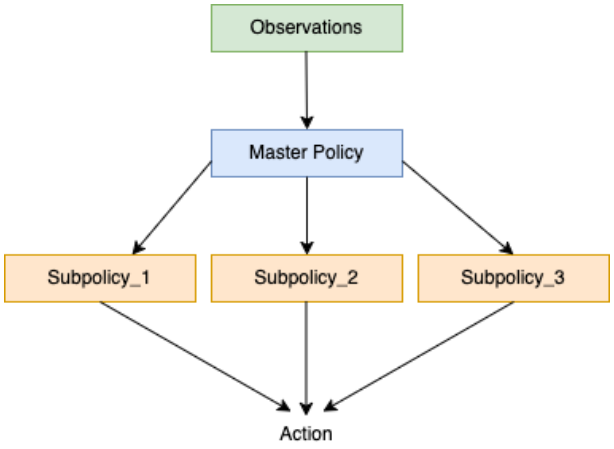


Figure 5: Hierarchical MARL.

to determine each agent's individual contribution to the overall reward. This is known as the **credit assignment** problem.
These challenges highlight the complexity of MARL and underscore the need for carefully designed algorithms and architectures to ensure effective learning and coordination among agents.

2.2.1 Decentralised Partially Observable Markov Decision Process (Dec-POMDP)

A decentralised partially observable Markov decision process (Dec-POMDP) models a multi-agent system in which agents must cooperate under partial observability. A Dec-POMDP is formally defined as a tuple

$$(\mathcal{S}, \{\mathcal{A}_i\}_{i\in I}, P, r, \{\Omega_i\}_{i\in I}, O, \gamma),$$

where:

- $I = \{1, \ldots, m\}$: the set of agents.
- $\mathcal{A}_i$: the finite action space for agent $i \in I$. the joint action space is $\mathcal{A} = \prod_{i\in I} \mathcal{A}_i$, and a joint action is denoted by $a \in \mathcal{A}$.
- $P(s' \mid s, a)$: the state transition probability of reaching s' from s when taking joint action a.
- $r(s, a, s')$: the shared reward received by all agents when transitioning from s to s' under joint action a.
- Ω_i: the finite observation set for agent $i \in I$. the joint observation space is $\Omega = \prod_{i\in I} \Omega_i$, and a joint observation is denoted by $o \in \Omega$.
- $O(o \mid s', a)$: the joint observation function; the probability of receiving joint observation o after taking joint action a and reaching state s'.

This framework is used in our research, as we are dealing with a multi-agent scenario.

2.3 Proximal Policy Optimisation (PPO)

In this section, we explain Proximal Policy Optimisation (PPO) in detail, as it is the foundation of all the algorithms used in this study. The algorithm was introduced by Schulman et al. [25]. PPO is a policy gradient method that samples steps from the environment and optimizes an objective function using stochastic gradient ascent. Unlike traditional policy gradient algorithms, which perform a single gradient update per data sample, PPO uses multiple minibatches over several epochs, making its training procedure more similar to supervised learning.

Compared to Trust Region Policy Optimisation (TRPO) [23], which defines a trust region in policy space based on KL divergence, PPO simplifies the Optimisation process by either clipping the objective function or adding a penalty term. This reformulation allows the use of standard first-order gradient ascent, resulting in significantly faster training on GPUs and easier implementation.

In this paper, we employ the most commonly used variant, clipped PPO. This method introduces a clipping mechanism in the objective function to restrict the policy update within a specified region, thereby discouraging the new policy from diverging too far from the old one. This leads to more stable training and improved sample efficiency.

PPO is an on-policy algorithm—meaning the policy used to collect data is the same as the one being improved, so it relies on fresh data generated by the current policy for each update rather than reusing old experiences—but it can be applied in both discrete and continuous action spaces, adding to its versatility. The pseudocode for PPO-Clip from OpenAI is shown in Figure 6.

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

Figure 6: Pseudocode for PPO-Clip [22].

The PPO algorithm uses two networks: the *Actor network* and the *Critic network*. The former selects actions from a probability distribution over all possible actions and the latter estimates the value of a particular state. In the beginning the policy parameters θ_0 and value-function parameters ϕ_0 are initialised with random weights.
For each training iteration, we run the current policy on the environment and collect batches of samples which will be used to update our networks. Then the advantage estimates are calculated, mostly using Generalized Advantage Estimation (GAE) [24]. The formula of the equation is presented in Equation 1.

$$\hat{A}_t^{\mathrm{GAE}(\lambda)} = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \, \delta_{t+l} \tag{1}$$

where:

- γ is the discount factor,
- λ is the GAE parameter controlling bias-variance tradeoff,
- δ_{t+l} is the temporal-difference (TD) residual at time $t + l$,
- T is the length of the trajectory or episode.

To clarify the roles of λ and δ, consider a single state transition $S_t \to S_{t+1}$:

$$\delta_t \;=\; r_t \;+\; \gamma\, V_\phi(s_{t+1}) \;-\; V_\phi(s_t)$$

Here, $V_\phi(S_t)$ is the critic (the NN used in PPO) and represents the predicted remaining return from state S_t. Updating with a single δ_t gives low variance but high bias, because it relies heavily on this imperfect critic.

At the other extreme, the Monte-Carlo return estimates the value by averaging the returns observed after visiting a state. This has very low bias but very high variance, reflecting the randomness of an entire episode.

Generalized Advantage Estimation (GAE) strikes a balance by forming an exponentially weighted running sum of future TD-errors. Each additional look-ahead step l contributes another TD-error whose influence is reduced by $(\gamma\lambda)^l$. When $\lambda \approx 0$, GAE approaches TD(0), relying mostly on the current critic; when $\lambda \approx 1$, it approaches the Monte-Carlo estimate, relying less on the critic and more on observed future rewards.

The intuition behind GAE is to estimate how much better or worse than expected that action turned out after looking λ-steps into the future, smoothing away just enough noise to make stable gradient steps without drowning in bias.
After that step, the policy update occurs where its formula is defined in Equation 2

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|\,T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \underbrace{\min\Big(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_k}(a_t \mid s_t)}\, A^{\pi_{\theta_k}}(s_t,a_t),\; g\big(\varepsilon, A^{\pi_{\theta_k}}(s_t,a_t)\big)\Big)}_{\mathcal{L}_t^{\mathrm{clip}}(\theta)} \tag{2}$$

$\mathcal{D}_k$**:** The batch of trajectories (rollouts) collected by executing the current policy π_{θ_k} once through the environment.

T**:** The number of time-steps in each trajectory.

π_θ**,** π_{θ_k}**:** π_θ is the *candidate* policy we are optimising; π_{θ_k} is the *behaviour* policy that generated the data.

Probability ratio

$$r_t(\theta) \;=\; \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_k}(a_t \mid s_t)}$$

quantifies how much the new policy changes the probability of the sampled action a_t relative to the old policy. For small updates $r_t(\theta) \approx 1$.

Advantage

$$A^{\pi_{\theta_k}}(s_t, a_t)$$

estimates how much better (or worse) the action a_t is compared with the average return from state s_t under the current policy. Positive advantages should be *encouraged*, negative ones *discouraged*.

Clipping function

$$g\big(\varepsilon, A\big) \;=\; \begin{cases} (1+\varepsilon)\, A & \text{if } A > 0, \\ (1-\varepsilon)\, A & \text{if } A < 0, \end{cases}$$

where the hyper-parameter $\varepsilon \in (0, 1)$ sets the width of the *trust region* (how far away the new policy can go from the old while still profiting the objective).

Clipped surrogate loss $\mathcal{L}_t^{\mathbf{clip}}(\theta)$ For better explanation let's investigate what happens when we have a single state-action pair (s, a) and break down the cases:

1. **Advantage is positive:** The contribution to the objective becomes

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon)\right) A^{\pi_{\theta_k}}(s, a).$$

   Since the advantage is positive, the objective will increase if the value of $\pi_\theta(a|s)$ also increases. The min operator puts a limit of how much the objective can increase. When $\pi_\theta(a|s) > (1+\epsilon)\pi_{\theta_k}(a|s)$, then it becomes $(1+\epsilon)A^{\pi_{\theta_k}}(s, a)$, therefore the new policy does not benefit by going far away from the old policy.

2. **Advantage is negative:** The contribution to the objective becomes

$$L(s, a, \theta_k, \theta) = \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1-\epsilon)\right) A^{\pi_{\theta_k}}(s, a).$$

   Since the advantage is negative, the objective will increase if the value of $\pi_\theta(a|s)$ decreases. The max operator puts a limit of how much the objective can increase. When $\pi_\theta(a|s) < (1-\epsilon)\pi_{\theta_k}(a|s)$, then it becomes $(1-\epsilon)A^{\pi_{\theta_k}}(s, a)$, therefore the new policy does not benefit by going far away from the old policy.

Overall objective We average $\mathcal{L}_t^{\text{clip}}(\theta)$ over all time-steps and trajectories, then maximise it (via stochastic gradient ascent with an optimiser such as Adam). This yields the next set of policy parameters θ_{k+1}.

The last step of the algorithm is the *Critic* (Value function) update. The formula is presented in Equation 4.

$$\phi_{k+1} = \arg\min_{\phi} \mathcal{L}^{\text{VF}}(\phi) \tag{3}$$

$$\mathcal{L}^{\text{VF}}(\phi) = \frac{1}{|\mathcal{D}_k|\,T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \big(V_\phi(s_t) - \hat{R}_t\big)^2, \tag{4}$$

$V_\phi(s_t)$**:** A neural network parameterised by ϕ that outputs an estimate of the expected return from state s_t.

$\hat{R}_t$**:** Target return for step t, the GAE target.

$\mathcal{L}^{\mathbf{VF}}(\phi)$**:** Mean-squared-error loss that measures how well the critic predicts the empirical returns.

Its role is to provides a low-variance baseline so the actor's advantages are centred, improving learning stability. Because the critic is trained on-policy each iteration, its estimate remains accurate for the data distribution the actor will see next.
All the steps described above are repeated across multiple iterations k until there is convergence.

2.4 Independent PPO (IPPO)

When dealing with multiple agents, as in this research, it is necessary to use algorithms designed for such settings. One such algorithm is Independent Proximal Policy Optimisation (IPPO), which operates under a decentralised training and decentralised execution framework [8]. IPPO does not require information sharing between agents, which simplifies both the training and execution processes. Each agent maintains its own policy and learns independently from local observations, without accessing any information from the other agents. In Figure 7, we illustrate the training process where each agent updates its policy only based on its own experiences and reward signals. This approach provides a lot of scalability and becomes effective in many MARL environments.

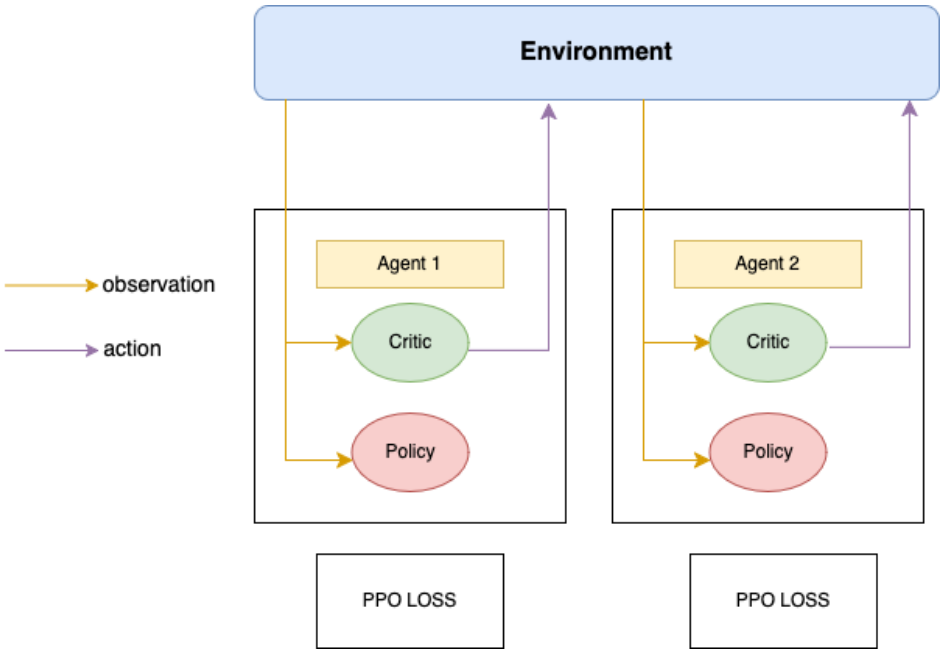


Figure 7: Illustration of the Independent Proximal Policy Optimisation (IPPO) framework.

2.5 Multi-Agent PPO (MAPPO)

MAPPO shares many similarities with IPPO and is also built on top of PPO, but it operates under a Centralised Training with Decentralised Execution [39]. This is achieved by using a centralised critic for all agents during training. The centralised critic receives global state observations from all agents and produces a shared value function. In Figure 8, we illustrate the flow of information from the environment to the shared critic. In the diagram each agent maintains its own policy, but they share one common critic. In certain cases, this approach can improve performance and coordination among agents.

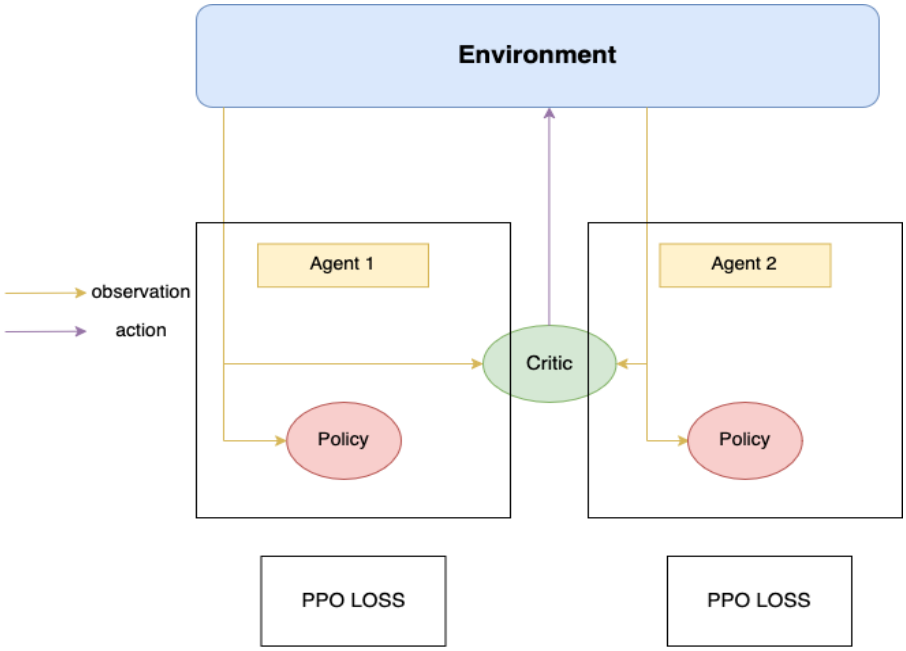


Figure 8: Illustration of the Multi-Agent PPO (MAPPO) framework.

2.6 CybORG Environment

In this subsection, a further analysis of our environment will be undertaken and its features and functionalities will be explained [1]. The environment is specifically designed for a series of four challenges, with the present study's exclusive focus being on the most recent iteration of the CAGE Challenge 4 [32]. Its purpose is to provide to researchers a cyber security research environment, thereby facilitating the training and evaluation of autonomous agents. A common interface is provided for emulated and simulated network environments. In order to make the environment more accessible, a variety of wrappers are available to allow for more flexible implementations with other libraries and frameworks. These challenges have evolved in terms of complexity and realism, with the CAGE challenge 4 focusing on a Multi-Agent Reinforcement Learning (MARL) scenario. This section aims to equip readers with a deep understanding of the environment, enabling them to effectively follow subsequent sections.

2.6.1 Environment Structure

The environment consists of a base station, subdivided into four main networks: two Deployed Networks (Network A and Network B), a Headquarters (HQ) Network, and a Contractor Network. These networks communicate through an internet connection. As illustrated in Figure 9, Network A and Network B each comprise two security zones: a restricted zone and an operational zone. The Headquarters zone is composed of three security zones: a Public Access Zone, an Admin Zone and an Office Network and the Contractor network only contains a single UAV control zone. Each security zone contains a random number of hosts and servers which ranges from 1-6 for servers and 3-10 for user hosts. Each host and

server has a minimum of one service and a maximum of five.
Finally, it is important to highlight that, consistent with previous discussions, most multi-agent environments are modelled as Dec-POMDPs, (including CybORG). The subsequent subsections will elaborate further on the specific components constituting CybORG's Dec-POMDP, as outlined in Section 2.

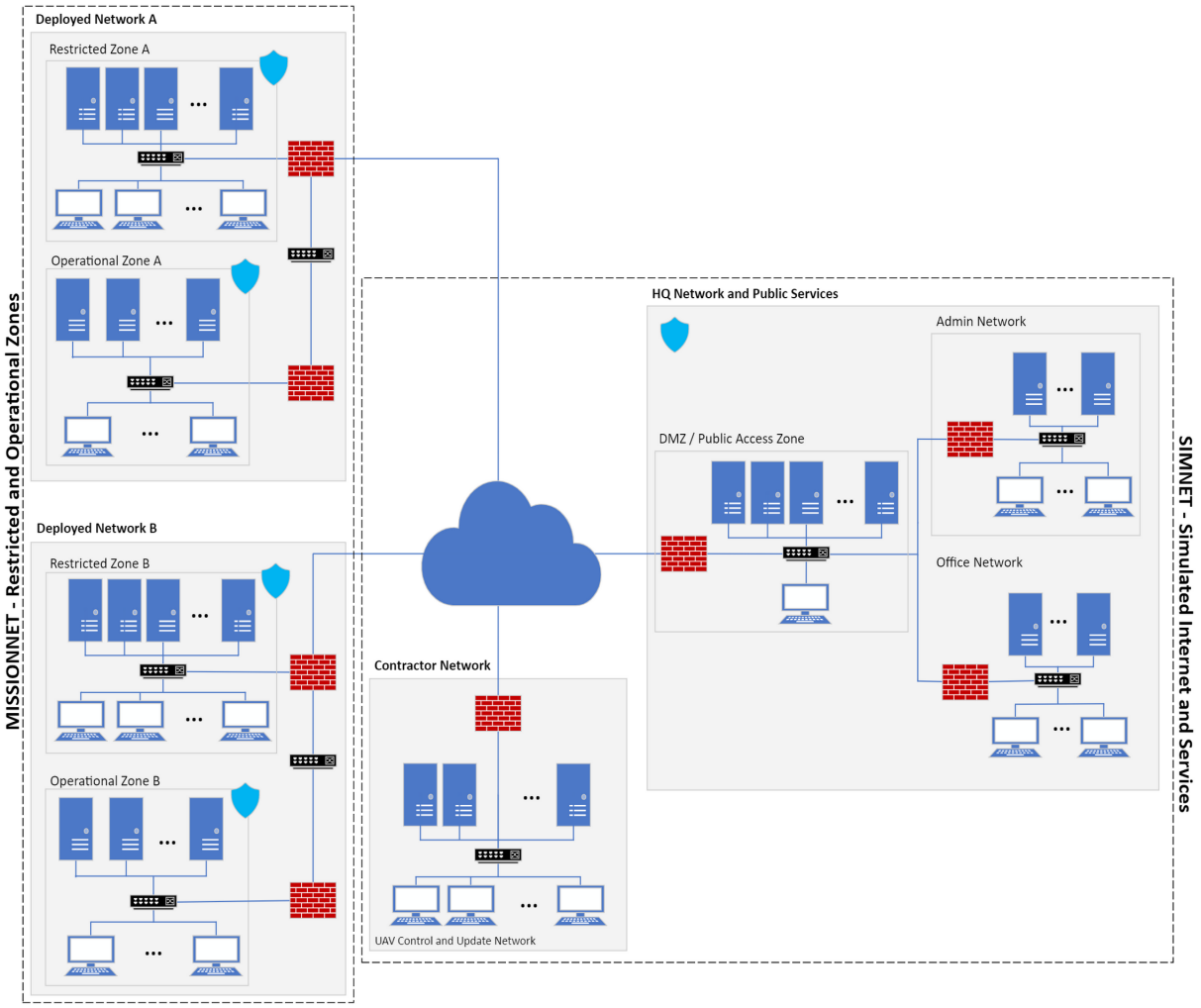


Figure 9: Network architecture from the CAGE Challenge 4 repository [32]. The diagram shows the four main networks, their internal zones and the five blue shields representing the locations of the blue agents.

2.6.2 Mission Phases

In this simulated scenario there are three main phases. The general priorities of each phase is to:

- Ensure the availability of critical network infrastructure to preserve essential operational capabilities.

- Maintain the servers responsible for day-to-day operations.
- Maintain uninterrupted access to public services.

Phase 1: The first phase focuses on maintaining general operations and prepare for major threats. There is no priority given over a particular mission in each zone, therefore there is a need for a balanced defence mechanism amongst the defenders. In Table 1 we can observe the network communication security policy in phase A as provided on the challenge's page. To better understand the communications between the different zones and networks, we also created a visual representation shown in Figure 10.

Phase 2A: The second phase focuses in Restricted Zone A and Operational Zone A which become mission critical. A key change in communication policy is that Operational Zone A becomes completely isolated and no longer connected to any other zone or network. Additionally, Restricted Zone A is now only connected to the HQ Network. This phase of the mission is very critical for the two blue agents in Deployed Network A, as they must defend their zones under these constrained communication conditions. The communication policies are presented in Table 2, with a corresponding visual representation shown in Figure 11.

Phase 2B: The third and final phase closely mirrors Phase 2A, but shifts the operational focus to Deployed Network B, which consists of Restricted Zone B and Operational Zone B. In this phase, Operational Zone B is completely disconnected from other networks and Restricted Zone B maintains connection only to HQ. Zones in Network A return to low priority and resume normal connectivity. The same level of pressure and responsibility seen in Phase 2A now transfers to the agents defending Network B. The communication policies for this phase are detailed in Table 3, with a corresponding visual representation provided in Figure 12.

Zone	HQ Network	Contractor Network	Restricted Zone A	Operational Zone A	Restricted Zone B	Operational Zone B	Internet
HQ Network	1	1	1	0	1	0	1
Contractor Network	1	1	1	0	1	0	1
Restricted Zone A	1	1	1	1	1	0	1
Operational Zone A	0	0	1	1	0	0	0
Restricted Zone B	1	1	1	0	1	1	1
Operational Zone B	0	0	0	0	1	1	0
Internet	1	1	1	0	1	0	1

Table 1: Phase A Communication Policy

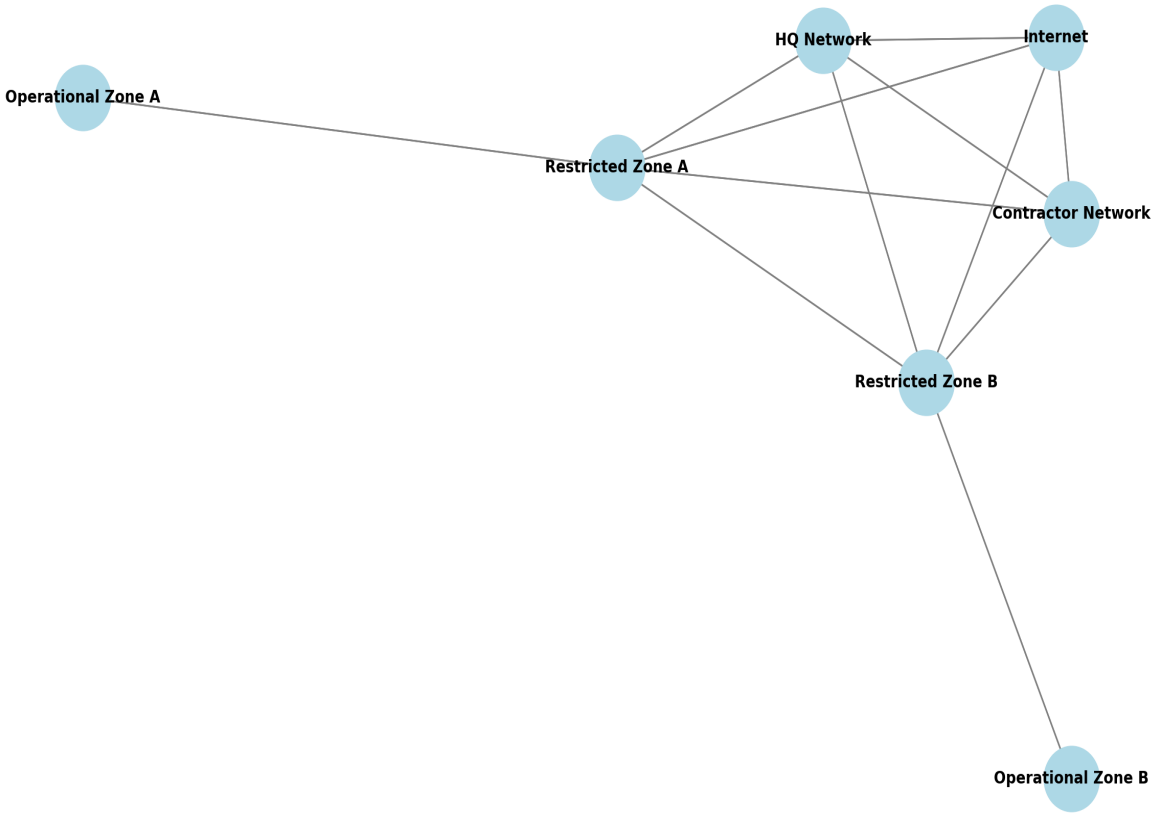


Figure 10: Phase A Communication Policy Graph.

Zone	HQ Network	Contractor Network	Restricted Zone A	Operational Zone A	Restricted Zone B	Operational Zone B	Internet
HQ Network	1	1	1	0	1	0	1
Contractor Network	1	1	0	0	1	0	1
Restricted Zone A	1	0	1	0	0	0	0
Operational Zone A	0	0	0	1	0	0	0
Restricted Zone B	1	1	0	0	1	1	1
Operational Zone B	0	0	0	0	1	1	0
Internet	1	1	0	0	1	0	1

Table 2: Phase 2A Communication Policy

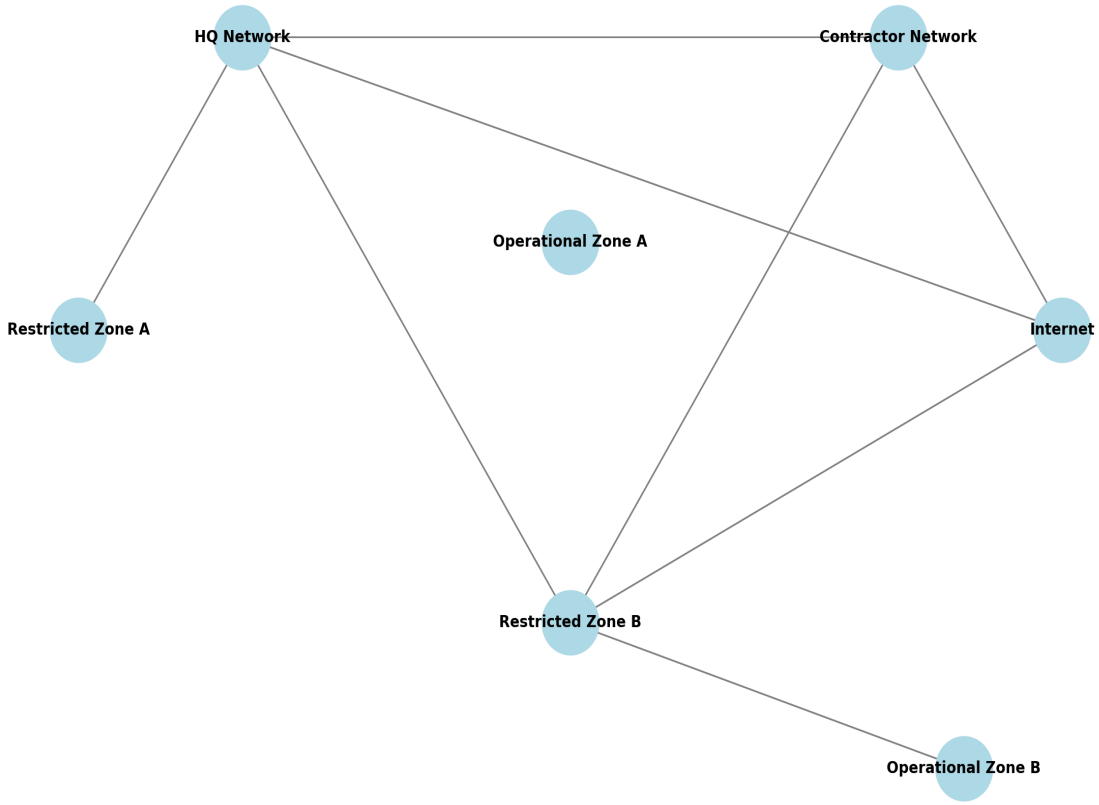


Figure 11: Phase 2A Communication Policy Graph.

Zone	HQ Network	Contractor Network	Restricted Zone A	Operational Zone A	Restricted Zone B	Operational Zone B	Internet
HQ Network	1	1	1	0	1	0	1
Contractor Network	1	1	1	0	0	0	1
Restricted Zone A	1	1	1	1	0	0	1
Operational Zone A	0	0	1	1	0	0	0
Restricted Zone B	1	0	0	0	1	0	0
Operational Zone B	0	0	0	0	0	1	0
Internet	1	1	1	0	0	0	1

Table 3: Phase 2B Communication Policy

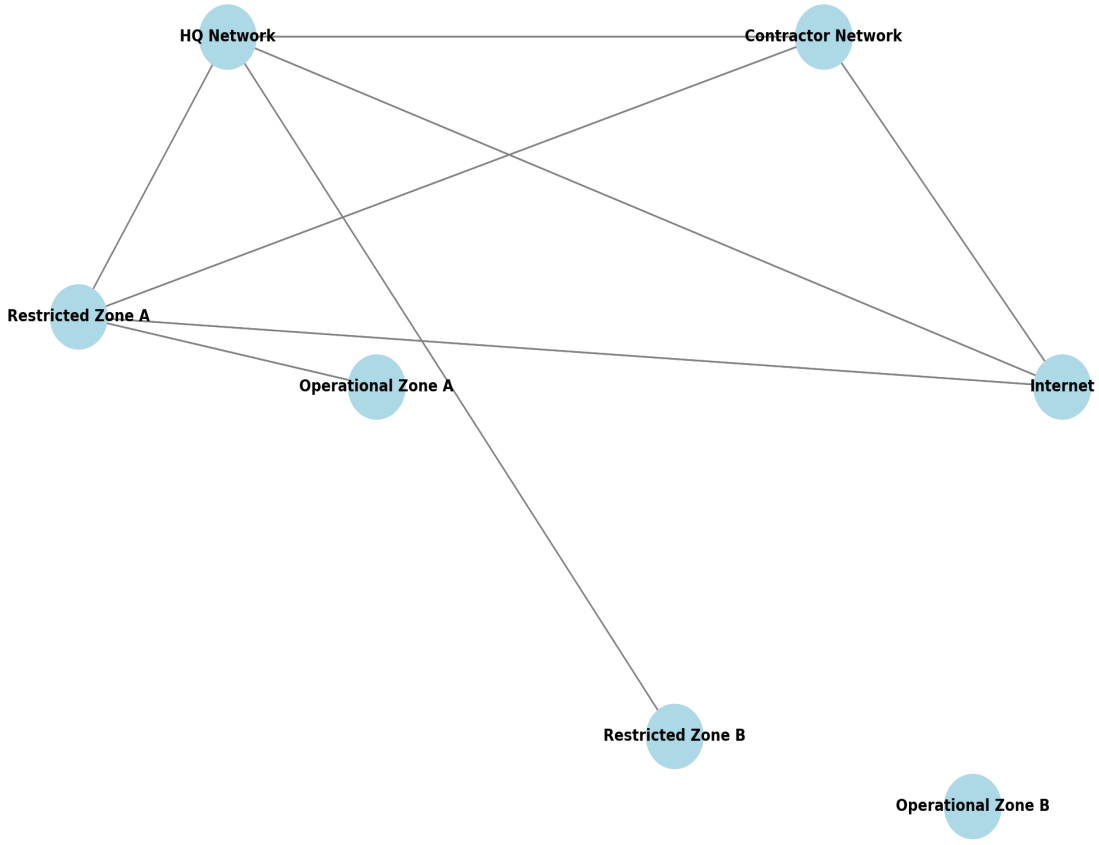


Figure 12: Phase 2B Communication Policy Graph.

2.6.3 Blue Agents (Defenders)

The base has 5 network defenders who are responsible for maintaining the security and functionality of their assigned networks. Networks A and B both have two blue agents, HQ network has one blue agent and the contractor network is undefended. They are assessed based on their ability to mitigate penalties associated with compromised services or failed user (green agent) tasks. Since each phase has different priorities as explained above, blue agents have to adapt to changing mission phases and evolving communication constraints. Each agent has a discrete action space as presented in Table 4 and each action requires a certain amount of time, measured in 'ticks' to execute.

Action	**Description**
`Monitor`	Default passive action that scans the network for malicious activity. Automatically performed when no other action is selected.
`Analyse`	Investigates a specific host to confirm or refute the presence of malware following an alert.
`DeployDecoy`	Deploys decoy services on hosts to lure red agents. Exploiting a decoy reveals red presence without risk to real services.
`Remove`	Attempts to eliminate malicious processes and files from a compromised host.
`Restore`	Restores a host to a known good state. This guarantees cleanup but temporarily disrupts green agent activity.
`BlockTraffic`	Blocks network traffic between specific subnets to isolate threats. May result in penalties if it prevents legitimate green traffic.
`AllowTraffic`	Reopens blocked network routes, enabling communication between zones.

Table 4: Available Actions for Blue Agents

2.6.4 Red Agents (Attackers)

Red agents try to compromise the network with the aim of disrupting operational services. They begin by randomly spawning to a random machine in the contractor network and attempt to pivot across the environment by exploiting vulnerabilities and navigating through allowed communication channels. In each zone there is a maximum of one red agent allowed, but these agents can maintain presence on multiple hosts. More red agents can be introduced during the episode, as a result of green agent behaviour, such as opening phishing emails or interacting with compromised services. Like blue agents, they also have a discrete

action space which is defined in Table 5 and employ randomly assigned strategies to mimic varying attack techniques. These actions are based on the MITRE ATT&CK® Framework to simulate real-world scenarios [17].

Action	Description
`DiscoverRemoteSystems`	Scans the network to identify additional hosts and IP addresses using ping sweeps.
`AggressiveServiceDiscovery`	Quickly scans a target host to identify open services. Has a high chance of triggering blue alerts.
`StealthServiceDiscovery`	Identifies services on a host more slowly but with a lower chance of being detected.
`ExploitNetworkServices`	Attempts to exploit a remote service to gain access to a host. Success depends on service vulnerabilities.
`PrivilegeEscalate`	Escalates privileges on a compromised host, enabling more powerful actions.
`DegradeServices`	Intentionally disrupts a host, increasing failure rates for green agent actions.
`Impact`	Executes actions that stop services or deny operations on the target host (e.g., destructive malware).
`DiscoverDeception`	Probes a service to determine whether it is a decoy planted by blue agents.
`Withdraw`	Removes the red agent from a host to avoid detection. Useful for stealth strategies.
`Sleep`	Takes no action for the current tick. Often used to delay or reduce exposure.

Table 5: Available Actions for Red Agents

2.6.5 Green Agents (Users)

Green agents are present in every host and can do local in local zones or remote work through accessing remote zones. When performing an action, the service and zone are selected randomly but have to obey the communication policy in each phase. Green agents are not directly controllable by the learning agents but play a crucial role in shaping the reward structure for blue agents. Penalties are applied when green agents are unable to complete their tasks, such as when their host is unavailable, or when they are blocked from accessing necessary services. The penalties are much harsher in mission-critical zones as described above. Another functionality is that they can generate false alerts by behaving in ways that resemble malicious activity. A summary of the two green agent actions is presented in Table 6.

Action	Description
`GreenLocalWork`	The green agent performs routine work on its local host without requiring any external communication. This action may occasionally result in red agent infiltration due to phishing or poor security practices.
`GreenAccessService`	The green agent attempts to connect to a service, either locally or in another valid zone, based on the current communication policy. Failures result in penalties if services are inaccessible or compromised.

Table 6: Available Actions for Green Agents

2.6.6 Rewards

At the beginning of each episode, blue agents start with an initial reward of 0. They receive penalties when green agents fail to complete their tasks, when they access compromised devices, or when a red agent executes the *Impact* action. In Phase 1, the reward structure is relatively balanced compared to later phases, with particular emphasis on the Contractor Network, which incurs significantly higher penalties of -5, as shown in Table 7. Phase 2A shifts the focus to Operational Zone A, resulting in a substantially higher penalty of -10 in that zone, as illustrated in Table 8. A similar pattern occurs in Phase 2B, where Operational Zone B becomes the focus, also receiving the highest penalty of -10 (Table 9). Theoretically, the maximum achievable reward in this environment is 0, which occurs when blue agents successfully and proactively prevent all red agent attacks.

Zone	Local Work Fails	Access Service Fails	Red impact/access
HQ Network	-1	-1	-3
Contractor Network	0	-5	-5
Restricted Zone A	-1	-3	-1
Operational Zone A	-1	-1	-1
Restricted Zone B	-1	-3	-1
Operational Zone B	-1	-1	-1
Internet	0	0	0

Table 7: Rewards for green action failures and compromise in Phase 1 O&M

2.6.7 Observations

In CAGE Challenge 4, the observation space is carefully structured to support multi-agent reinforcement learning. Each agent operates autonomously with the possibility of communicating with another agent according to the communication policy of each phase. Each agent receives relevant, local information, which is critical in this complex and partially observable environment.

Zone	Local Work Fails	Access Service Fails	Red impact/access
HQ Network	-1	-1	-3
Contractor Network	0	0	0
Restricted Zone A	-2	-1	-3
Operational Zone A	-10	0	-10
Restricted Zone B	-1	-1	-1
Operational Zone B	-1	-1	-1
Internet	0	0	0

Table 8: Rewards for green action failures and compromise in Phase 2A - Mission A

Zone	Local Work Fails	Access Service Fails	Red impact/access
HQ Network	-1	-1	-3
Contractor Network	0	0	0
Restricted Zone A	-1	-3	-3
Operational Zone A	-1	-1	-1
Restricted Zone B	-2	-1	-3
Operational Zone B	-10	0	-10
Internet	0	0	0

Table 9: Rewards for green action failures and compromise in Phase 2B - Mission B

Each observation vector is composed of three main components: the mission phase, information about the subnets that each agent is responsible for using a structured block of features and four message blocks that are received from the other agents. As illustrated in Figure 13, the general representation of the agents' observation space is demonstrated.

Figure 13: State representation of the CybORG environment for an agent, consisting of Mission Phase (MP), Subnetworks (S1, S2, ..., SF), and Message vector (M).

3 Related Works

The field of RL for Cyber-Defence has seen a surge in research activity in recent years, with significant progress being made in this area. However, it should be noted that the results obtained thus far are still considered to be of an experimental nature.

Studies that focus exclusively on defence against cyber-physical systems (CPS), incorporate various RL algorithms to develop defence mechanisms, demonstrating their effectiveness in controlled environments. Whilst certain approaches emphasise the disruption of attacker models [19], other focus on the identification of vulnerabilities [4]. The applications span a variety of CPS contexts, including as IoT systems, smart grids, and general cyber-physical infrastructures, demonstrating RL's adaptability in tackling a wide range of security issues.

An analysis of papers focusing on Denial of Service Attacks (DoS) reveals a variety of methodologies employed. Although most of the studies use MARL [3] in addressing complex challenges across different technological landscapes, studies such as Simpson et al. [26] introduce a novel approach to per-host DDoS mitigation utilising direct-control RL with two classes of RL agents.

This study will focus exclusively on the deployment of RL agents for the purpose of defending against network cyber-attacks. A survey of the existing literature reveals that the majority of papers found in defence of network attacks employ MARL, as most network scenarios are composed of subnetworks in which each agent is responsible for managing and defend a specific subnetwork. In particular, the paper by Cardelline et al. [6] employs RL agents to determine optimal responses to intrusions detected in a web application. The authors of the paper in question are cognisant of the non-stationary nature of the task, a fact which distinguishes their work from some other studies that make the assumption that the structure of the protected system does not change over time [11]. This is the only study found that applies the knowledge gained in simulated environments to real-world applications. In addition, other studies have explored various RL techniques for analogous tasks in other simulated environments [36] [35], achieving optimal performance outcomes for their specific environment. Wiebe et al.[35] trained agents within cyMARL, an extension of the CybORG simulator [29] to collaboratively detect and mitigate attacker activities. It employs different methodologies comparing Independent Learning and Centralised Training with Decentralised Execution (CTDE). This study is closely aligned with our research, as it also utilises the CybORG environment. Additionally, several studies employing the CybORG environment have adopted significant simplifications that reduce the complexity of the simulation, resulting in scenarios that require only a single reinforcement learning agent [14, 37]. Another approach, distinct from those in other studies, enables agents to develop and adapt strategies on the fly while allowing defenders to share their experiences, thereby enhancing overall efficiency [40]. Although agents can share experiences, they are currently unable to directly transfer their knowledge of defence mechanisms, a restriction that could be explored in future studies. One of the most important limitations of all papers, is the ability to test the models in a range of real-world applications. The approaches facilitated in each research show promising results, but there is a pressing need for more realistic emulators or simulators to bridge the gap between experimental outcomes and real-world applications [36]. Moreover, the manual tuning of hyperparameters necessitates a sophisticated approach,

which is of paramount importance in order to achieve optimal results.

In the present study, a comparison of MARL algorithms will be conducted with the objective of optimizing performance and attaining high rewards within the simulated environment. For this part, we will build upon a Master's thesis and investigate its top-performing algorithms, placing greater emphasis on hyperparameter tuning [7]. Subsequently, the performance of Hierarchical MARL will be evaluated in comparison to that of traditional structures, in order to determine whether the former offers superior results. An existing research paper that employs hierarchical MARL methods in the CybORG environment will serve as the foundation for this part of our study [27].

4 Reinforcement Learning Methods

In this section, we present and analyse the reinforcement learning methods used in our implementation, which are closely related to those discussed in Section 2, within the context of the CybORG environment.

4.1 Global State Representation

In the IPPO implementation, each agent receives its own local observations in the form of a vector, as shown in Figure 13. This procedure is the same for both training and execution phases. In contrast, the MAPPO implementation uses a centralised critic during training which has access to the local observations of all agents (often referred to as global state). However, during execution, the MAPPO agents operate only based on their own local observations. Figure 14 illustrates how this concatenated state vector is fed into the centralised critic. For simplicity, the activation functions in the neural network (NN) are omitted in the illustration. Additionally, the figure shows each neuron receiving an entire observation vector per agent. In reality, since each observation is a vector, each neuron receives a single value from that vector. Therefore, the actual input to the neural network has a dimensionality of $x \cdot n$, where x is the size of each observation vector and n is the number of agents. Our NN has two hidden layers, each consisting of 256 neurons. The output is a single scalar representing the predicted value of the state.

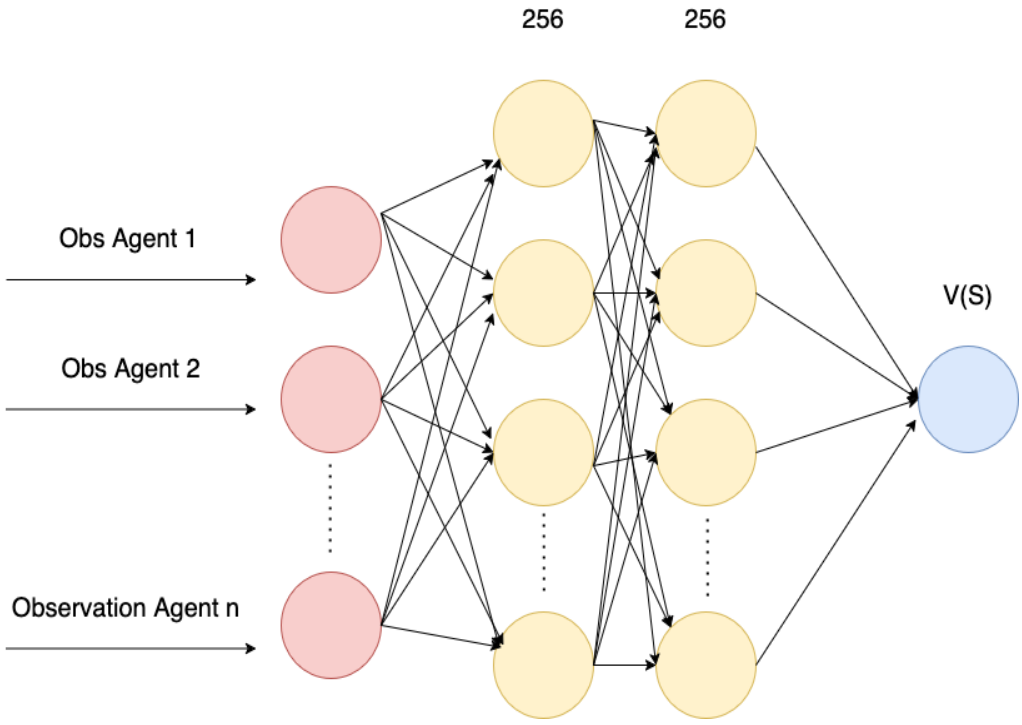


Figure 14: Architecture of the MAPPO central critic during the training phase.

4.2 Hyperparameter tuning protocol

In machine learning, selecting appropriate hyperparameters is crucial for stable and effective learning. This is particularly important when using PPO, as it involves several tunable parameters that can significantly impact the agent's performance. Given that training is time-consuming (typically taking around 15 to 25 hours for 200 training iterations) we opted to tune a subset of hyperparameters by sampling from different values. These include the clip ratio, batch size, minibatch size, and learning rate. Following common practices in RL, we chose these specific hyperparameters as they have the most significant impact on stability and performance [2, 13] . The other hyperparameters were kept fixed, and the values are shown in Table 11.

To facilitate hyperparameter optimisation, we used the Optuna library. Specifically, we employed the Tree-Structured Parzen Estimator (TPE) sampler [34], a Bayesian optimisation method. Intuitively, this algorithm focuses the search on regions that are likely to yield good results while avoiding regions that are likely to perform poorly, based on prior evaluations. We also used the multivariate version of TPE, in which the good and bad parameter distributions are modeled jointly using multivariate Gaussian mixtures across the full parameter space. This allows the sampler to learn and exploit correlations between hyperparameters more effectively.

Additionally, we integrated the Asynchronous Successive Halving Algorithm (ASHA) [15], an aggressive early stopping strategy designed for large-scale hyperparameter tuning in parallel computing environments. In our experiments, we used a grace period of 5 (the minimum number of iterations a trial had to run before it could be stopped) and a maximum of 100 or 200 iterations, depending on the specific experiment. We used the average episode return to optimise our scheduler.

To answer our first research question, we first established a baseline configuration using the values shown in Table 10. We then compared subsequent trials to see whether better performance could be achieved.

Hyperparameter	**Value**
Batch size	100 000
Minibatch size	4 000
Clip Ratio (per agent)	0.3
Learning rate (per agent)	5e-5

Table 10: Hyperparameter settings for the baseline experiments.

For IPPO, we split the hyperparameter tuning into five trials per algorithm, as we were unable to secure a GPU for a single, extended training session. The resulting learning curves are shown in Figure 17 in Section 5.1. In each trial, five sampling runs were conducted for 100 iterations. As illustrated, some runs are shorter than others—either due to early stopping or because the allocated GPU session expired, resulting in an abrupt halt to training. To ensure we obtained the best hyperparameters, we selected the three best-performing runs and retrained using those specific hyperparameters for 200 iterations.

For MAPPO, we repeated the IPPO protocol but extended training to 200 iterations to assess whether doubling the number of iterations improved performance. However, the experiment was constrained by frequent CPU out-of-memory errors and GPU-node timeouts. As a result, we completed only four trials with a single sampling run each—except for one trial, which included two sampling runs. The results are shown in Figure 18.

Parameter	
Discount factor (γ)	0.99
GAE λ	1
Network size	[256, 256]
Optimizer	Adam
Target KL divergence	0.01
Episode update interval	30 episodes
Entropy coefficient (per agent)	0.001

Table 11: Fixed hyperparameters for IPPO and MAPPO in the CybORG environment.

4.3 Hierarchical Structure

In Figure 5, we presented the abstractions of Hierarchical Multi-Agent Reinforcement Learning (HMARL). In this subsection, we explain how this structure was applied to our environment, based on the methodology proposed by Singh et al. [27]. We employed one master policy and two subpolicies, as illustrated in Figure 15.

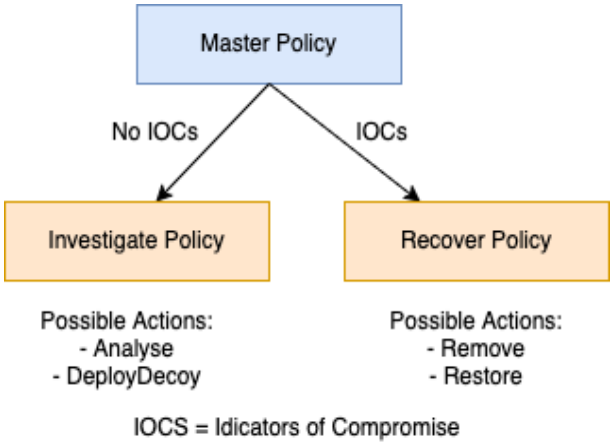


Figure 15: HMARL policies.

The master policy is responsible for selecting a specific subpolicy based on the agent's observation. The chosen subpolicy then selects one of the available actions. This hierarchical structure can lead to more efficient and targeted exploration, potentially resulting in improved performance.

For the Investigate subpolicy, the blue agent has two available actions: analyzing a specific host or setting up a decoy on a specific host. These actions are chosen because there is no

clear indication of compromise, so the blue agent focuses on detecting red-agent activity. In contrast, the Recover subpolicy also offers two actions: attempting to remove a red agent from a host or restoring the network to a known good state. These actions are taken when there are signs of compromise, prompting the blue agent to respond accordingly.

To train the policies, we employed a deterministic rule for the master policy. A deterministic rule was chosen to simplify the training process, without introducing additional complexities of training a third, higher level policy. As illustrated in Figure 15, if indicators of compromise (IOCs) are detected, the Recover subpolicy is selected; otherwise, the Investigate subpolicy is used. Each subpolicy was then trained independently using PPO.

As shown in Table 4, two additional actions (Block Traffic and Allow Traffic) were available but excluded from our simulation. These actions usually result in negative outcomes for the blue agent and do not lead to increased rewards because they can disrupt the actions of green agents, resulting in a negative reward.

For HMARL training, we again employed the baseline values listed in Table 10 to verify the effectiveness of the training. Due to resource limitations, we carried out only two runs instead of the four performed for IPPO and MAPPO. The results are presented in Section 5.2.

Because the HMARL results were unstable, we decided to try additional hyperparameters to see whether performance would improve. We carried out two more runs with the values presented in Table 12. The idea behind these changes is to enable slower but more stable learning. A smaller learning rate can help avoid overshooting local optima and reduce the risk of divergence. Higher batch and minibatch sizes typically reduce the variance of the gradients and lead to more stable updates. Finally, a lower clip ratio prevents large changes in the policy in a single step, which also improves training stability.

Hyperparameter	**Value**
Learning rate	1e-5
Batch size	150 000
Minibatch size	6 000
Entropy coefficient	0.001
Clip ratio	0.2

Table 12: Hyperparameter settings for additional HMARL runs

5 Results and Discussion

In this section, we present the results for the two research questions outlined in Section 1. First, we analyse the performance of our two main algorithms across various hyperparameter settings. Second, we compare the hierarchical methods against the traditional approaches. For our baseline figures in this section, we compute the mean return at each episode across all trials and display a 95% confidence interval around those means. This approach quantifies the uncertainty of our estimates: the shaded region between the upper and lower bounds should, in principle, contain the true average return curve approximately 95% of the time. A narrower band corresponds to greater confidence in our results. The tables below the figures show the set of hyperparameters used in each experiment.

5.1 Research Question 1

Our first research question concerns the optimal hyperparameters for the IPPO and MAPPO algorithms. To enable a fair comparison, we established a baseline using the default values listed in Table 10. Agents were trained in the CybORG environment with a different random seed for each trial, and baseline results were averaged over four runs per algorithm to improve robustness. Each episode comprised 500 timesteps.

The corresponding learning curves are shown in Figure 16. Training spanned approximately $20,000,000$ timesteps, and the final average return per episode was nearly identical for the centralised critic (MAPPO) and the individual critics (IPPO), with MAPPO performing slightly better than IPPO. Because this pattern is consistent across all random seeds, we are confident that MAPPO performs better than IPPO. Both algorithms converged around -1500 average return per episode, with MAPPO attaining a slightly higher final mean return (-1490 ± 20) than IPPO (-1510 ± 20). .

Figure 17 shows the average return per episode for several hyperparameter settings for IPPO. Some trials outperform the baseline even with just 100 iterations; however, because their batch sizes are larger, each iteration covers more timesteps. The most promising configuration was the green curve in Trial 3, whose final average return was approximately -1450. In Figure 19, we can see that the three best performing trials achieve a bit better performance within 200 iterations, but they all converge around the -1350 ± 50 mark.

Figure 18 illustrates the average return per episode for several hyperparameter settings for MAPPO, constrained by the limitations outlined in Section 4.2. Despite testing only a small number of configurations, the results are encouraging and suggest that suitable hyperparameters can markedly improve the algorithm's performance. In particular, Trial 1 attains a final average return of approximately -1200.

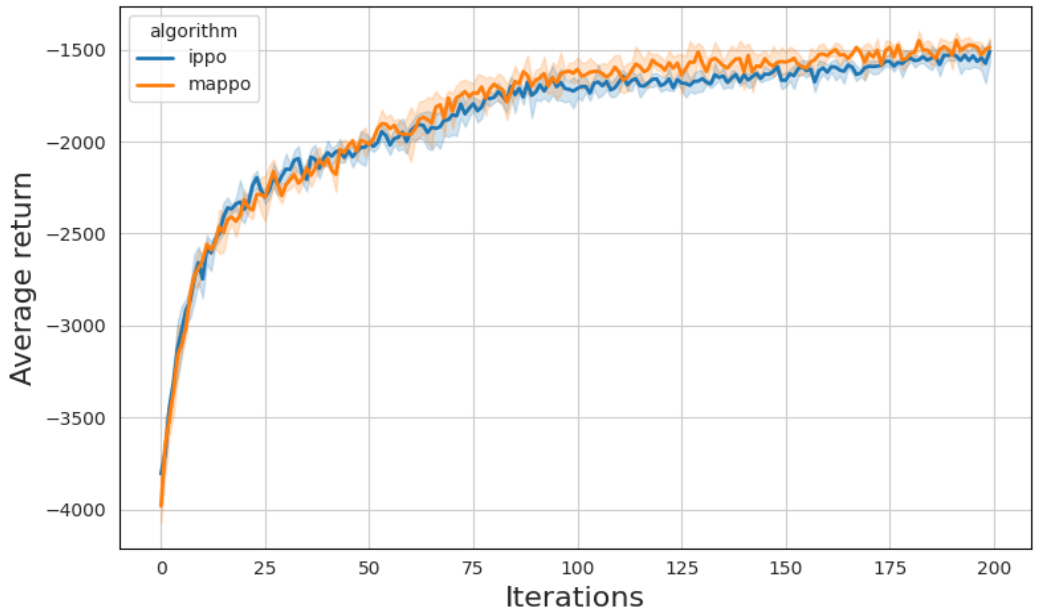
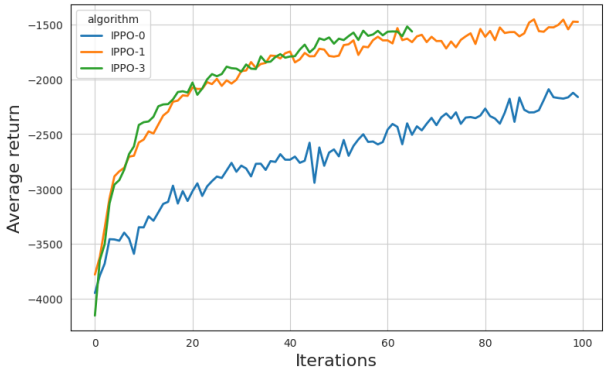


Figure 16: Baseline result obtained from using IPPO and MAPPO algorithms in the baseline CybORG environment. MAPPO shows a slight and consistent performance advantage over IPPO.

It is noteworthy that our results contradict those reported by Davide [7]: we observe that MAPPO outperforms IPPO. Intuitively, this makes sense because the agents learn from each other's observations and rewards, fostering better cooperation and coordination. In addition, the centralised critic provides a more stable value-function estimate, leading to more reliable policy updates and faster learning.

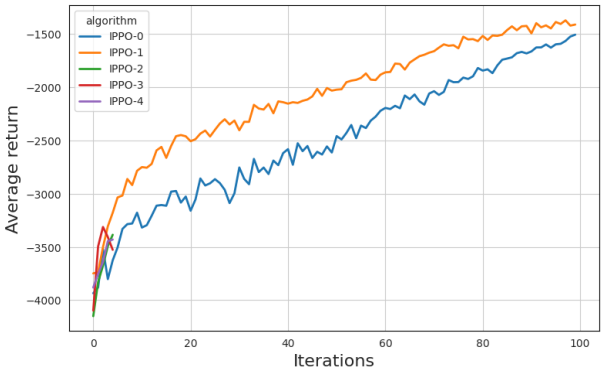
Regarding our first research question, we have shown that hyperparameter tuning improves the performance of both PPO algorithms, and that exploring a wider range of values could yield further gains. As for which algorithm performs better, MAPPO consistently outperforms IPPO, showing slightly superior results in the baseline configuration and across all tuned trials.

We believe that larger batch sizes yield more stable gradient updates and significantly improve the average return per episode. Resource constraints, however, prevented us from testing even larger batch sizes. Moreover, several learning curves (especially in the IPPO experiments) had not converged when training stopped, as indicated by rewards that continued to rise in the final iterations. This suggests that training for more iterations could further improve performance and perhaps achieve a higher average return per episode. Nevertheless, although these algorithms yield substantial improvements, they still fall short of achieving truly strong performance (e.g., a return of –150 per episode as shown in the CAGE Challenge 4 results leaderboard). This suggests that alternative methods will be required to fully solve this environment.



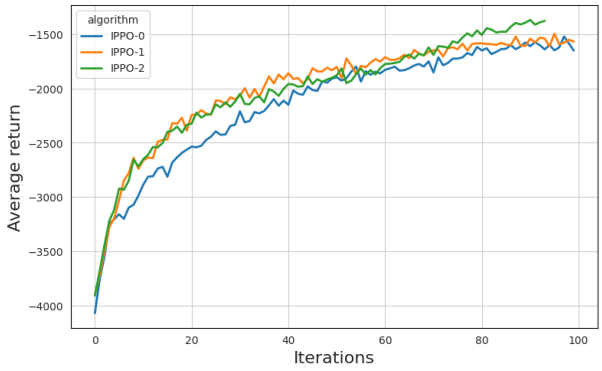
(a) Trial 1

Trial	Learning Rate	Clip	Batch Size	Minibatch Size
IPPO-0	2.47×10^{-4}	0.2025	132,169	7,145
IPPO-1	2.14×10^{-5}	0.2710	167,949	2,849
IPPO-3	4.55×10^{-5}	0.2391	188,130	6,286



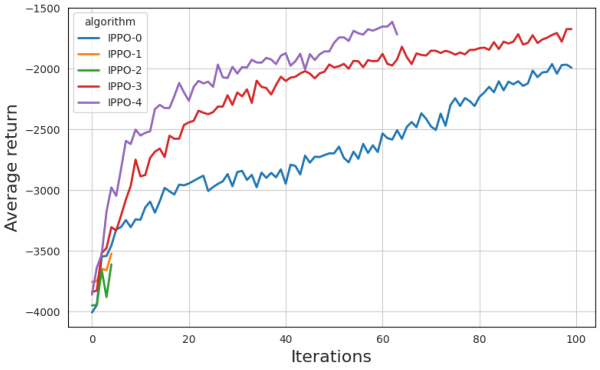
(b) Trial 2

Trial	Learning Rate	Clip	Batch Size	Minibatch Size
IPPO-0	1.24×10^{-4}	0.1191	136,447	3,488
IPPO-1	5.30×10^{-5}	0.1743	153,846	3,253
IPPO-2	3.94×10^{-5}	0.1159	126,779	5,236
IPPO-3	1.31×10^{-4}	0.2581	127,763	2,912
IPPO-4	1.12×10^{-4}	0.2828	117,981	2,071



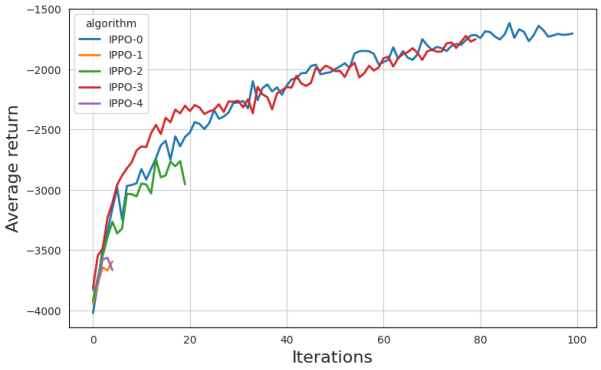
(c) Trial 3

Trial	Learning Rate	Clip	Batch Size	Minibatch Size
IPPO-0	1.47×10^{-5}	0.1816	140,350	4,211
IPPO-1	2.84×10^{-5}	0.2636	132,076	2,980
IPPO-2	4.06×10^{-5}	0.1757	177,718	3,889



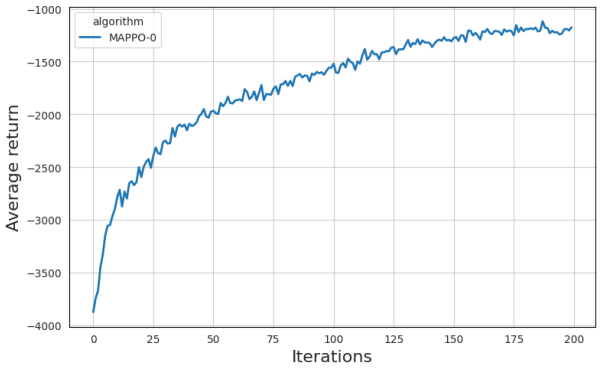
(d) Trial 4

Trial	Learning Rate	Clip	Batch Size	Minibatch Size
IPPO-0	1.87×10^{-4}	0.1803	183,553	4,425
IPPO-1	2.10×10^{-4}	0.2376	110,298	4,865
IPPO-2	2.08×10^{-4}	0.1168	127,543	2,520
IPPO-3	1.88×10^{-5}	0.1064	173,651	2,899
IPPO-4	3.94×10^{-5}	0.2768	132,497	2,913



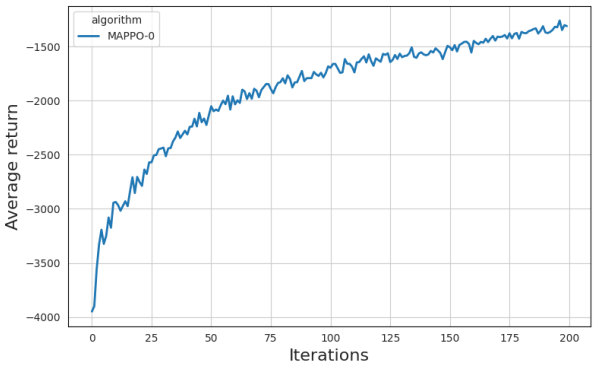
(e) Trial 5

Trial	Learning Rate	Clip	Batch Size	Minibatch Size
IPPO-0	1.96×10^{-5}	0.2815	109,709	4,071
IPPO-1	1.36×10^{-4}	0.1089	183,528	2,438
IPPO-2	1.43×10^{-4}	0.1896	169,867	4,884
IPPO-3	4.54×10^{-5}	0.2881	118,063	2,399
IPPO-4	2.19×10^{-5}	0.2464	102,482	7,862



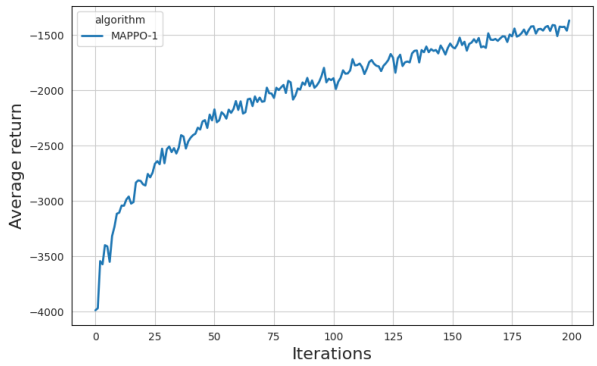
(a) Trial 1

Trial	Learning Rate	Clip	Batch Size	Minibatch Size
MAPPO-0	1.53×10^{-5}	0.1957	132,668	3,392



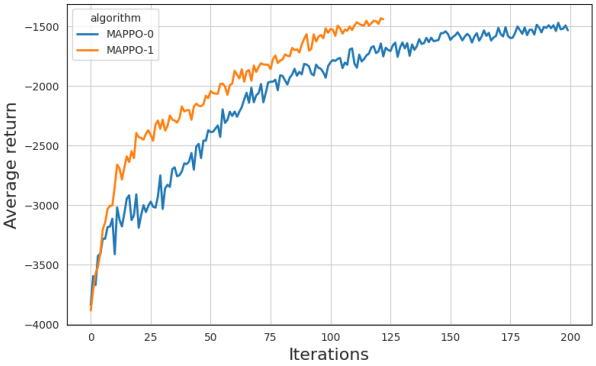
(b) Trial 2

Trial	Learning Rate	Clip	Batch Size	Minibatch Size
MAPPO-0	1.33×10^{-5}	0.1574	125,261	4,507



(c) Trial 3

Trial	Learning Rate	Clip	Batch Size	Minibatch Size
MAPPO-1	1.68×10^{-5}	0.1122	111,148	6,177



(d) Trial 4

Trial	Learning Rate	Clip	Batch Size	Minibatch Size
MAPPO-0	1.26×10^{-4}	0.2083	113,346	3,390
MAPPO-1	6.49×10^{-5}	0.1363	129,250	7,912

Figure 18: Average return per episode for four MAPPO hyperparameter trials.

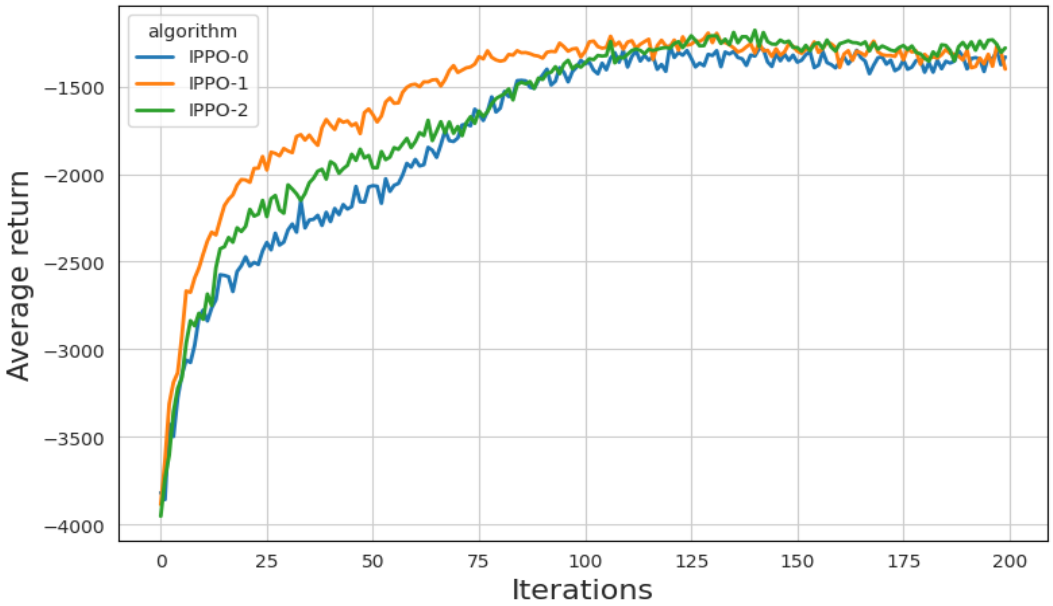


Figure 19: Average return per episode for the best 3 IPPO trials in the baseline CybORG environment. All the trials outperform the default baseline (Figure 16), converging to a final average return of approximately -1350 ± 50.

Trial	Learning Rate	Clip	Batch Size	Minibatch Size
IPPO-0	5.30×10^{-5}	0.1743	153,846	3,253
IPPO-1	4.55×10^{-5}	0.2391	188,130	6,286
IPPO-2	4.06×10^{-5}	0.1757	177,718	3,889

5.2 Research Question 2

Our second research question investigates whether employing HMARL and leveraging specialised subpolicies together with primitive actions can improve agent learning. The baseline results shown in 20 were averaged over two runs, with each episode consisting of 500 timesteps.

Comparing this curve with those of IPPO and MAPPO, we observe that all three methods rise steeply during the first 25 iterations. HMARL plateaued at roughly -2000 ± 100 average return per episode, while MAPPO and IPPO continued to improve gradually and converge at approximately -1500 ± 50. Another notable difference is the initial performance: MAPPO and IPPO start near -4000 average return per episode, whereas HMARL begins around -7000 yet climbs far more rapidly, indicating strong early-stage sample efficiency.

After about 25 iterations, the HMARL curve ceases to improve and displays repeated oscillations. This behaviour may arise from re-using hyperparameters tuned for flat MARL algorithms or from unresolved credit-assignment challenges in the hierarchical architecture.

As the baseline hyperparameters performed suboptimally, Figure 21 shows the results obtained using the hyperparameters described in Table 12. We observe some improvement

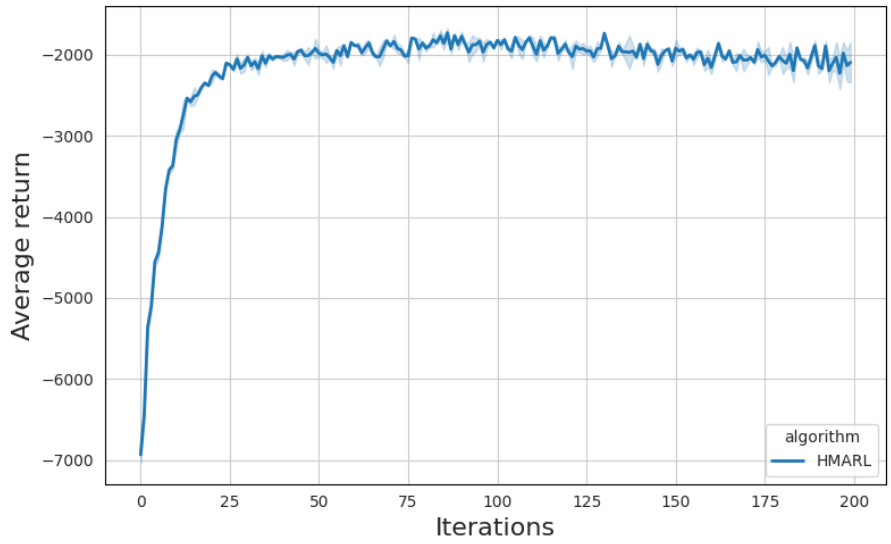


Figure 20: Baseline result obtained from using HMARL algorithm in the baseline CybORG environment. The initial learning is much faster but converges to a suboptimal final average return in comparison with the traditional algorithms.

compared to the baseline result, with a final average return of around −1800. Another interesting finding is that the return improves gradually over the 200 iterations, in contrast to the curve in Figure 21, where the average return decreases after around 125 iterations. This suggests that the algorithm could be further improved, perhaps by using larger batch sizes and more iterations to aid the learning process.

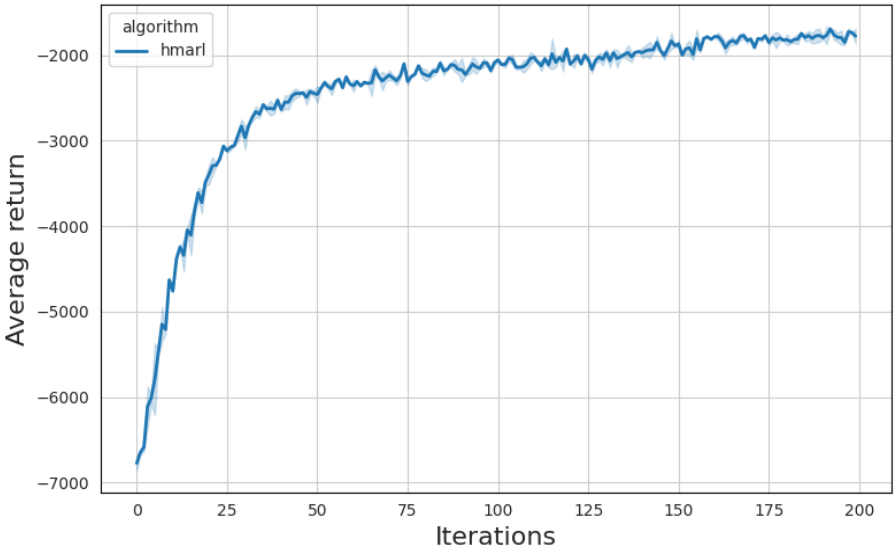


Figure 21: Result obtained from using HMARL algorithm in the baseline CybORG environment with the different hyperparameters (Table 12). The final average return is a bit higher but again not optimal compared to the traditional algorithms.

In summary, HMARL quickly captures the environment's structure and learns rapidly in the early stages-an advantage when training time is limited. Under the current settings its asymptotic performance remains roughly 400 return points below that of IPPO and MAPPO. We believe that a targeted hyperparameter optimisation, particularly of the learning rate schedule and entropy regularisation, could substantially improve its long run performance. Table 13 shows the best rewards obtained during training using each of the three methods employed in this study. Figure 22 displays the best runs from each training method together for easier comparison. As expected, MAPPO and IPPO produce very similar results, converging to around -1200 ± 50, whereas HMARL's performance is much worse, converging to around -1800 ± 20.

Algorithm	Best Average Return
MAPPO	-1117
IPPO	-1191
HMARL	-1667

Table 13: Best mean episodic return by algorithm during training.

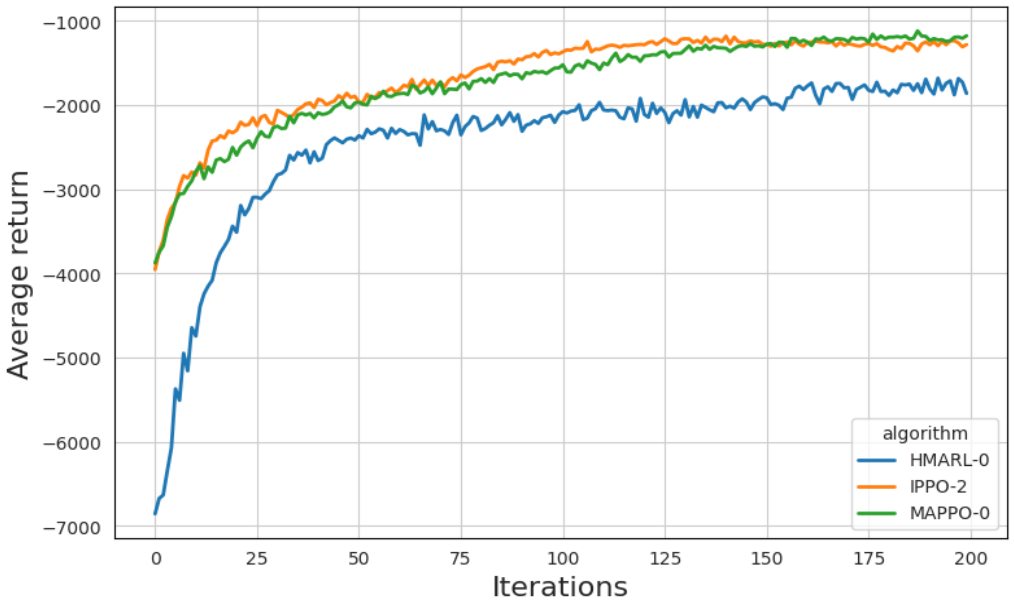


Figure 22: Average return over best training iterations for HMARL, IPPO, and MAPPO algorithms using their respective hyperparameters.

Trial	Learning Rate	Clip	Batch Size	Minibatch Size
HMARL-0	1.00×10^{-5}	0.3000	150,000	6,000
IPPO-2	4.06×10^{-5}	0.1757	177,718	3,889
MAPPO-0	1.53×10^{-5}	0.1957	132,668	3,392

6 Conclusion

This study aims to investigate major RL methods in the context of cybersecurity to determine their effectiveness in simulated scenarios. Our results showed that MAPPO outperforms IPPO in all experiments in this environment and that tuning the hyperparameters of standard algorithms is crucial for their performance, but more experiments are needed to test additional values. Furthermore, the HMARL results were not optimal and performed worse than the standard MARL algorithms. The initial learning was faster but converged to a lower asymptotic result in the end.

Compared to other state-of-the-art findings in CAGE Challenge 4, our results were not optimal. However, from open-source research, we found that many studies used other methods, including hardcoding, to take advantage of specific observations in the environment. While this is efficient for a particular environment, we believe that a more generalised approach to real-time attacks would be more robust in the future.

Future work could proceed in several directions. First, we have shown that hyperparameter tuning affects PPO performance. By exploring a wider range of values may yield further gains. Second, the environment could be modified (e.g. experimenting with alternative message-passing schemes, reward structures, or other tweaks that enhance information flow and agent performance). Finally, other RL algorithms could be tested to see whether they offer additional improvements.

7 Acknowledgments

I would like to thank my supervisor, Dr. Fatih Turkmen, for giving me the opportunity to work with him on my bachelor's thesis and for guiding me towards the final result. I would also like to thank my second supervisor, Rafael Cunha, for helping me with some practical implementation details and for giving me advice on technical ideas. Finally, I would like to thank Davide Rigoni, whose master's thesis inspired my work. He was also very helpful in answering all my questions and was always eager to help.

References

[1] Cyber operations research gym. https://github.com/cage-challenge/CybORG, 2022. Created by Maxwell Standen, David Bowman, Son Hoang, Toby Richer, Martin Lucas, Richard Van Tassel, Phillip Vu, Mitchell Kiely, KC C., Natalie Konschnik, Joshua Collyer.

[2] Jacob Adkins, Michael Bowling, and Adam White. A method for evaluating hyperparameter sensitivity in reinforcement learning, 2025.

[3] Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024.

[4] Mohammed Alhajri, Houbing Song, Mohamed Amine Ferrag, Leandros Maglaras, and Helge Janicke. Security analysis of cyber-physical systems using reinforcement learning. *Sensors*, 23(3):1634, 2023.

[5] Tom Bewley. Markov decision processes. https://gibberblot.github.io/rl-notes/single-agent/MDPs.html, 2020.

[6] V. Cardellini, E. Casalicchio, S. Iannucci, M. Lucantonio, S. Mittal, D. Panigrahi, and A. Silvi. An intrusion response system utilizing deep q-networks and system partitions. *SoftwareX*, 19:101120, 2022.

[7] Davide236. Collaborative rl cage 4. https://github.com/Davide236/Collaborative_RL_CAGE_4, 2023.

[8] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip H. S. Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge?, 2020.

[9] Neil Dhira, Henrique Hoeltgebaum, Niall Adams, Mark Briers, Anthony Burke, and Paul Jones. Prospective artificial intelligence approaches for active cyber defence, 2021. arXiv:2104.09981v1.

[10] Martin Eling and Werner Schnell. Cyber risk and cybersecurity: A systematic review of data availability. *The Geneva Papers on Risk and Insurance - Issues and Practice*, 47(3):698–736, 2022.

[11] Bingrui Foo, Yu-Sung Wu, Yu-Chun Mao, Saurabh Bagchi, and Eugene Spafford. Adepts: Adaptive intrusion response using attack graphs in an e-commerce environment. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2005.

[12] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3–4):219–354, 2018.

[13] Jacob Hilton, Karl Cobbe, and John Schulman. Batch size-invariance for policy optimization, 2022.

[14] Mitchell Kiely, David Bowman, Maxwell Standen, and Christopher Moir. On autonomous agents in a cyber defence environment. *arXiv preprint arXiv:2309.07388*, 2023.

[15] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning, 2020.

[16] Yunjie Liu, Jianhua Zhang, Qiang Liu, Wei Wang, and Wei Zhang. Cyber-security and reinforcement learning — a brief survey. *Engineering Applications of Artificial Intelligence*, 109:105116, 2022.

[17] MITRE. MITRE ATT&CK Framework. https://attack.mitre.org/, 2024.

[18] Reza Montasari. Machine learning and deep learning techniques in countering cyberterrorism. In *Cyberspace, Cyberterrorism and the International Security in the Fourth Industrial Revolution: Threats, Assessment and Responses*, pages 135–158. Springer International Publishing, Cham, 2024.

[19] Sayak Mukherjee and Veronica Adetola. A secure learning control strategy via dynamic camouflaging for unknown dynamical systems under attacks. In *2021 IEEE Conference on Control Technology and Applications (CCTA)*, pages 905–910, 2021.

[20] Sanghyun Oh, Jeongyoon Kim, and Jongyoul Park. Employing deep reinforcement learning to cyber-attack simulation for enhancing cybersecurity. *Electronics*, 13(3):555, 2024.

[21] Emmanuel Ok. Ai-powered malware analysis: A comparative study of traditional vs. ai-based approaches. https://www.researchgate.net/publication/383395000_AI-Powered_Malware_Analysis_A_Comparative_Study_of_Traditional_vs_AI-Based_Approaches, 2024. Preprint on ResearchGate.

[22] OpenAI. Proximal policy optimization. https://spinningup.openai.com/en/latest/algorithms/ppo.html, 2018.

[23] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[24] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[26] Kyle A. Simpson, Steven Rogers, and Dimitrios P. Pezaros. Per-host ddos mitigation by direct-control reinforcement learning. *IEEE Transactions on Network and Service Management*, 17(1):15–28, 2020.

[27] Aditya Vikram Singh, Ethan Rathbun, Emma Graham, Lisa Oakley, Simona Boboila, Alina Oprea, and Peter Chin. Hierarchical multi-agent reinforcement learning for cyber network defense. *arXiv preprint arXiv:2410.17351*, 2024.

[28] Edward J. Sondik. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978.

[29] Maxwell Standen, Martin Lucas, David Bowman, Toby J. Richer, Junae Kim, and Damian Marriott. Cyborg: A gym for the development of autonomous cyber agents. *arXiv preprint arXiv:2108.09118*, 2021.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

[31] Imad Tareq, Bassant M. Elbagoury, Sanaa A. El-Regaily, and El-Sayed M. El-Horbaty. Deep reinforcement learning approach for cyberattack detection. *International Journal of Online and Biomedical Engineering (iJOE)*, 20(5):15–30, 2024.

[32] CAGE Challenge Team. Cage challenge 4 - network diagram. https://github.com/cage-challenge/cage-challenge-4, 2024.

[33] Shria Verma. Redefining network security: A comparative study of traditional and ai-driven approaches. *International Journal of Creative Research Thoughts (IJCRT)*, 11(12):567–574, 2023.

[34] Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance, 2023.

[35] Jacob Wiebe, Ranwa Al Mallah, and Li Li. Learning cyber defence tactics from scratch with multi-agent reinforcement learning. *arXiv preprint arXiv:2310.05939*, 2023.

[36] Alec Wilson, Ryan Menzies, Neela Morarji, David Foster, Marco Casassa Mont, Esin Turkbeyler, and Lisa Gralewski. Multi-agent reinforcement learning for maritime operational technology cyber security, 2024.

[37] Melody Wolk, Andy Applebaum, Camron Dennler, Patrick Dwyer, Marina Moskowitz, Harold Nguyen, Nicole Nichols, Nicole Park, Paul Rachwalski, Frank Rau, and Adrian Webster. Beyond cage: Investigating generalization of learned autonomous network defense policies. *arXiv preprint arXiv:2211.15557*, 2022.

[38] J. Yi and X. Liu. Deep reinforcement learning for intelligent penetration testing path design. *Applied Sciences*, 13(16):9467, 2023.

[39] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games, 2022.

[40] Tianqing Zhu, Dayong Ye, Zishuo Cheng, Wanlei Zhou, and PS Yu. Learning games for defending advanced persistent threats in cyber systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 53(4):2410–2422, 2022.