Universiteit Leiden
The Netherlands

Bachelor Computer Science

Adversarial Honeypot Detection and Deployment with Reinforcement Learning

Valentijn Ouwehand

Dr.ir. E. Makri
Dr. T.M. Moerland

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 15/01/2026

Abstract

This thesis explores the application of Reinforcement Learning (RL) in the context of honeypot detection and deployment in a simulated network. Honeypots are deceptive systems designed to attract and engage attackers. However, the increasing use of sophisticated detection techniques presents new challenges. The goal of this thesis is to investigate the interaction between a Reinforcement Learning attacker and defender in a honeypot network. The attacker attempts to learn an optimal policy for detecting honeypots in a network, and the defender aims to find the optimal strategy for disguising these honeypots. Through a series of experiments in a simulated network environment, this research evaluates the interaction between these two agents. The findings suggest that RL-based strategies can enhance honeypot detection techniques, even when faced with an opposing defender agent.

Contents

1 Introduction

Cybersecurity continues to be an important area of research as the frequency and sophistication of cyberattacks increase. One strategy that has proven to be effective in defending against these threats is the use of honeypots. Honeypots are deceptive systems designed to attract and engage attackers by simulating vulnerable targets and diverting the attackers' attention. However, modern attack strategies enable more accurate detection of honeypots in computer networks. Techniques such as fingerprinting and behavioral analysis reduce the effectiveness of conventional static deployment strategies [MSA+22].

Dynamic and adaptive honeypot deployment strategies are needed to address this challenge. This is where machine learning, and specifically Reinforcement Learning (RL), can play an important role. RL allows an agent to make adaptive decisions based on feedback received from an environment. The ability of an RL-agent to learn from previously made mistakes can be a promising tool in the context of adaptive honeypot systems. For defenders, RL can help optimize the interaction behavior of honeypots, making them more difficult to detect. For attackers, RL offers a way to adaptively identify honeypots by learning from interactions with the network.

Even though RL has been explored in cybersecurity for various applications, there is limited research on its use for honeypot detection. This thesis aims to fill that gap by investigating how RL can be used for honeypot detection and defense. The attacker agent will learn how to detect honeypots using a dynamic policy, while the defender agent will adapt its honeypot interaction tactics to reduce the likelihood of detection. By studying the interactions between these agents, this research aims to explore how RL can be used to make honeypot systems more resilient to advanced detection techniques.

The thesis is structured as follows. Chapter 2 reviews related work and background on honeypots and Reinforcement Learning. Chapter 3 defines the problem statement and the specific research goals of this study. Chapter 4 details the research methodology, including the design of the experimental framework and the models used for both the attacker and defender agents. Chapter 5 describes the experimental setup, while Chapter 6 presents the experimental results. Chapter 7 discusses the implications of these results, followed by conclusions and suggestions for future work in Chapter 8.

All code used for the experiments and simulations in this thesis is publicly available[1].

[1] https://github.com/vaal2001/HoneypotAdverserial

2 Related Work & Background

2.1 Honeypots

Cyber deception is a well-established defense strategy [JJT^{+}24]. Honeypots are among the most studied and effective cyber defense techniques [ZT21], designed to attract attackers by imitating legitimate services and wasting their effort [FDFV17]. In essence, a honeypot is a deceptive system that emulates genuine vulnerabilities, deliberately designed to mislead attackers into believing they have compromised a legitimate target [FACU21]. In contrast with traditional intrusion detection systems, a honeypot intentionally presents itself as an interesting service in order to lure attackers, instead of passively observing real systems for signs of intrusion [JS11].

Honeypots are differentiated by their interaction level with the attacker, generally falling into low-, medium-, or high-interaction categories, each reflecting the degree of engagement an attacker can achieve [FACU21]. Low-interaction honeypots emulate only basic services without a full operating system, making them easy to deploy, low-risk, and resource-efficient, but they yield limited intelligence about potential attackers and are easier for these attackers to detect [IDSM23]. Conversely, high-interaction honeypots expose genuine system environments to attackers, capturing comprehensive behavioral and forensic data at the cost of greater operational risk and resource demand [WSDE09]. Medium-interaction honeypots are designed as a middle ground between low- and high-interaction systems, providing a higher degree of realism and engagement while maintaining lower operational risk and complexity [FACU21].

While honeypots are powerful tools for gathering behavioral and forensic data, they face growing challenges due to the increasing sophistication of detection techniques from the attackers, which undermine their effectiveness [URLL17]. Advanced attackers are increasingly using honeypot fingerprinting, a process employed to identify honeypots by detecting signs of emulation or controlled environments [SPV23]. These signs often reveal that the system is running within a virtual machine and include, but are not limited to, inconsistent protocol implementations, abnormal network and system configurations, and unnatural timing patterns [SPV23, FDFV17, JJT^{+}24]. Furthermore, the absence of typical user activities, such as logins, background processes, or regular network traffic, may suggest the presence of a honeypot [LWS23]. In response to these challenges, recent research has focused on developing anti-fingerprinting techniques, such as implementing more realistic interaction patterns and dynamically adjusting network and system behavior [JJT^{+}24]. These developments highlight the ever-evolving nature of the cyber security field, which continuously identifies new vulnerabilities and develops corresponding countermeasures [JJT^{+}24].

2.2 Reinforcement Learning

The content in this section is largely based on the foundational concepts of Reinforcement Learning (RL) as discussed by Sutton and Barto in their book *Reinforcement Learning: An Introduction* [SB18]. RL is a branch of machine learning focused on training agents to make decisions by interacting with an environment. The agent learns by receiving feedback in the form of rewards or penalties, depending on the actions it takes, with the aim of maximizing cumulative rewards over time. Unlike supervised learning, where the model is trained on labeled data, RL relies on trial and error, where agents continuously adjust their actions based on the outcomes to achieve long-term goals rather than immediate results.

At the core of RL lies the interaction between the agent and the environment. The agent makes decisions, selecting actions based on the current state of the environment. The environment reacts to these actions, providing feedback through rewards or penalties. A state represents a specific configuration of the environment at a given time, which informs the agent's decision-making process. After taking an action based on the current state, the agent receives feedback in the form of a reward. Positive rewards reinforce actions that move the agent closer to its goal, while negative rewards discourage actions that lead to undesired outcomes.

One of the central challenges in RL is the exploration vs. exploitation dilemma. Exploration involves trying new actions to discover potentially better strategies or hidden opportunities, while exploitation focuses on leveraging known actions that maximize rewards based on the agent's current knowledge. Balancing exploration and exploitation is crucial for an agent's learning process and long-term performance.

To guide its decision-making, an RL agent relies on a policy, a strategy that defines the action to take for each possible state. Over time, the agent improves its policy by learning from the feedback received from the environment. The value function is another key component in RL, which estimates the expected long-term reward for each state or action. It helps the agent evaluate the desirability of different states and select actions that will maximize its cumulative reward.

Several RL algorithms are commonly employed to train agents. A key distinction in Reinforcement Learning algorithms is between on-policy and off-policy methods.

On-policy algorithms are those where the agent learns the value of the policy that it is currently following. In other words, the agent improves its policy using the actions it is actually taking. One common on-policy algorithm is Proximal Policy Optimization (PPO), which directly optimizes the policy by adjusting the probabilities of actions based on the feedback received. PPO ensures that updates are stable and prevents large, destabilizing changes by employing a clipped objective function [SWD+17].

On the other hand, Off-policy algorithms learn from actions that are not necessarily being taken by the agent under its current policy. This means that the agent can learn from past experiences or actions taken by other agents. One such algorithm is Q-Learning, which learns the value of each action in a given state. Q-Learning uses this information to derive an optimal policy, but unlike on-policy methods, it can use experiences gathered from other exploration strategies or even from a random policy to improve its own decision-making.

Thus, the primary difference lies in the policy the agent uses to collect experiences: on-policy methods learn from actions taken by the current policy, while off-policy methods learn from actions taken by different policies, allowing for more flexibility in learning from past experiences.

2.2.1 Markov Decision Processes and Optimization Objective

Reinforcement Learning problems are commonly formalized as a Markov Decision Process (MDP). An MDP is defined as a tuple

$$(\mathcal{S}, \mathcal{A}, P, R, \gamma) \tag{1}$$

where $\mathcal{S}$ is the set of states describing the environment, $\mathcal{A}$ is the set of actions available to the agent, $P(s'|s, a)$ defines the transition probability from state s to state s' after taking action a, $R(s, a)$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards.

At each timestep t, the agent observes the current state s_t, selects an action a_t, receives a reward $r = R(s_t, a_t)$, and transitions to a new state s_{t+1} according to the environment dynamics.

The behavior of an agent is defined by a policy π, which specifies a probability distribution over actions given a state:

$$\pi(a|s) = \Pr(a_t = a | s_t = s). \tag{2}$$

In this thesis, stochastic policies are used, meaning actions are sampled from this distribution rather than selected deterministically.

The objective of the agent is to maximize the expected cumulative discounted reward, also referred to as the return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{3}$$

The state-value function $V^\pi(s)$ represents the expected return obtained when starting from state s and following policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t | s_t = s \right] \tag{4}$$

Similarly, the action-value function $Q^\pi(s, a)$ represents the expected return when taking action a in state s and subsequently following policy π:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t | s_t = s, a_t = a \right]. \tag{5}$$

The goal of Reinforcement Learning is to find an optimal policy π^* that maximizes the expected return over the distribution of initial states:

$$\pi^* = \arg\max_\pi \; \mathbb{E}_{s_0}[V^\pi(s_0)] \tag{6}$$

In practice, this optimization is performed using approximate methods such as policy gradient algorithms, which directly adjust the parameters of a policy to increase expected return. In later sections of this thesis, this objective is optimized using Proximal Policy Optimization (PPO).

2.3 Honeypots & Reinforcement Learning

In recent years, honeypots have become a well-established defense strategy in cybersecurity, particularly for attracting and deceiving attackers by simulating vulnerable systems [MDR25].

These deceptive systems are strategically placed within a network to mimic legitimate services and waste the attackers' time and resources. However, as attackers become more sophisticated, they are increasingly able to detect and bypass honeypots, making static placement strategies less effective [Iye21]. To address these challenges, there has been growing interest in leveraging Reinforcement Learning (RL) to enhance honeypot deployment and management [GZCLP25].

Reinforcement Learning (RL) has proven to be an effective tool for adaptive decision-making, and it is now being applied to enhance honeypot deployment and management. For example, Huang and Zhu [HZ19] used RL to model honeypot engagement using Semi-Markov Decision Processes (SMDPs). Unlike traditional Markov Decision Processes (MDPs), SMDPs allow for more flexibility by incorporating variable time durations between state transitions. This characteristic is particularly useful in scenarios where the time an agent spends in a state (such as engaging with an attacker) is not fixed but varies depending on the circumstances, like how long the attacker interacts with the honeypot [HZ19].

In their study, Huang and Zhu [HZ19] focused on key aspects of honeypot engagement, such as well time and risk-cost trade-offs. Dwell time refers to the amount of time an attacker spends interacting with a honeypot before discovering it as a decoy or deciding to disengage. By modeling this factor with RL, the agent can learn to optimize how long it keeps an attacker engaged, delaying their discovery of the honeypot and maximizing the time they waste interacting with the system. On the other hand, risk-cost trade-offs capture the balance between the cost of maintaining a honeypot (e.g., computational resources, risk of detection) and the potential benefits of deceiving the attacker. The RL model in Huang and Zhu's approach adapts the honeypot deployment strategy by weighing these trade-offs, helping to minimize the likelihood of detection while maintaining effective deception.

Additionally, Guan et al. [GLC+23] explored the use of RL for high-interaction honeypots in the Internet of Things (IoT) environment, where the RL agent learns to respond dynamically to attacker behavior. By adapting its responses based on real-time interactions, the RL-based honeypot improves its ability to deceive attackers, particularly by bypassing reconnaissance tools and providing more realistic responses that make the honeypot harder to detect. The study demonstrated how RL could be used to increase the realism of the honeypot, which is critical in environments where attackers are becoming more sophisticated and equipped with advanced detection mechanisms.

Further research, such as Deep Reinforcement Learning for Building Honeypots against Runtime DoS Attack [VK22], demonstrates the potential of RL in adapting honeypots to specific attack vectors, such as Denial of Service (DoS) attacks, by dynamically configuring honeypot settings to maintain their deceptive nature under attack conditions. Another relevant study by Li et al. [LZW25] utilizes RL to determine the optimal deployment strategy for multiple types of honeypots, which may include different interaction levels (e.g., low- vs. high-interaction honeypots) and functional categories (e.g., network, application, or database honeypots), under resource constraints. This further demonstrates RL's utility in managing complex honeypot configurations and adapting them to a variety of attack scenarios.

These studies highlight how RL can be applied to honeypot deployment by allowing the defender to adjust the placement and interaction strategies based on ongoing attacker activity. By using RL, defenders can optimize their honeypot placement, making the system more adaptive and harder to detect, thereby increasing its effectiveness.

While the application of RL to honeypot deployment has shown promising results, there is a notable gap in the literature regarding the use of RL for honeypot detection. The current research predominantly focuses on improving honeypot effectiveness from the defender's perspective [Iye21], but there is limited exploration of how RL can be used by attackers to identify honeypots. Most traditional detection techniques rely on static, signature-based methods, or simple heuristics like timing analysis and network behavior analysis [SPV21]. However, these methods can be circumvented by sophisticated attackers using more dynamic strategies.

There is currently limited research on how attackers could leverage Reinforcement Learning (RL) to learn and adapt their strategies for detecting honeypots. However, there has been growing interest in using machine learning and Reinforcement Learning techniques for adversarial purposes, such as evading detection systems. By training an RL-based attacker, it would be possible to simulate an evolving adversary capable of identifying honeypots through interaction patterns or subtle environmental cues, such as virtualization artifacts, protocol inconsistencies, or missing user activity. These characteristics are often present in honeypots but not in legitimate systems, and RL could allow attackers to identify these discrepancies dynamically over time. This concept is closely related to recent work where attackers use adversarial machine learning to adapt to changing defensive strategies, such as network traffic analysis or intrusion detection systems [LZL+25]. This would provide a dynamic and evolving challenge for defenders, as attackers would continuously adapt to new honeypot placement strategies and attempt to evade detection. For instance, Reinforcement Learning has been used in other security domains to dynamically learn how to bypass detection systems, making it a promising technique for attackers targeting honeypots as well [QMMF22]. In this thesis, the above mentioned research gap by developing a framework in which Reinforcement Learning (RL) is used to train an attacker agent capable of dynamically adapting its strategies to detect and bypass honeypots is addressed.

3 Problem Statement

Although Reinforcement Learning has been applied to improve defense strategies, there has been limited exploration of how it can be used by attackers to adaptively identify honeypots. Currently, honeypot detection techniques remain largely static.

This research aims to explore the interaction between attackers and defenders within a Reinforcement Learning framework. The central research question of this thesis is:

> **How effectively can an RL-based attacker detect honeypots, and how can an RL-based defender adapt deployment strategies to minimize detection?**

To address this question, three sub-questions will be explored:

1. How well can an RL-attacker learn to detect honeypots deployed randomly in a network?

2. How well can an RL-defender adaptively deploy honeypots to evade detection when faced with conventional detection heuristics such as signature-based or timing analyses?
3. How do interactions between RL-based attacker and defender agents evolve when trained simultaneously?

By investigating these questions, this research provides insights into the effectiveness of traditional honeypot strategies in countering emerging threats. The study contributes to the development of adaptive honeypot systems capable of adjusting to the evolving tactics of attackers in real time.

4 Research Methodology

4.1 Experimental Design Overview

The previous section provided an overview of the research direction. The main research question is supported by three sub-questions, each corresponding to a single atomic experiment. The results from these individual experiments contribute to answering the main research question.

The first sub-question, *"How effectively can an RL-based attacker learn to detect honeypots deployed randomly in a network?"*, investigates whether a baseline RL-based attacker to identify honeypots within a network exists. This experiment evaluates the attacker's ability to distinguish between real hosts and honeypots by reporting on its classification accuracy.

The second sub-question, *"How well can an RL-based defender adaptively deploy honeypots to evade detection when faced with conventional detection heuristics such as signature-based or timing analysis?"*, seeks to explore the development of a baseline RL-based defender that adaptively evades traditional detection heuristics. The defensive strategies is evaluated against conventional attack techniques and their ability to deceive the attacker is examined.

The agents designed in these first two experiments form the basis for the third and final sub-question: *"How do interactions between RL-based attacker and defender agents evolve when trained simultaneously?"* This final experiment focuses on the dynamic interactions between the two agents, with opposing goals, within a shared environment. The adversarial context provides insights into how each agent adapts in response to the other. Ultimately, the performances of both agents exposed to each other's strategies are compared.

4.2 Environment Specification

The performance of a Reinforcement Learning agent is strongly influenced by the environment with which it interacts [Res20]. In an ideal situation, the environment is composed of an accurate simulation of a real-life enterprise-level network. However, creating such an environment is computationally expensive and impractical. Therefore, abstractions must be made to simplify the environment, while maintaining enough realism to ensure that the agent can effectively learn and make decisions.

Previous research in the context of honeypot defenders indicates that there is no perfect abstraction level for simulating network environments [JJT+24]. The difficulty lies in finding a balance between computational feasibility and the richness of the environment's representation. For the purposes of this study, a graph-based model to represent the network environment is used. This allows the representation of relationships between devices in the network while maintaining computational efficiency.

4.2.1 Graph Design

A graph is a data structure that represents relationships between entities [Anu24]. In the context of a network, the graph consists of nodes, representing devices, and edges, representing the connections between the devices. This graph is a largely simplified representation of a real-world enterprise-level network. The nodes in the graph represent either a real device or a honeypot. Using this setup as the environment, the RL-based attacker agent is tasked to distinguish between the types of devices. Although the graph structure is an abstraction of the real network, the nodes can still accurately imitate the behavior of real devices and honeypots. This can be achieved through simulation or emulation [GKCS23]. Simulation, in the context of honeypots, models the behavior of hosts to resemble that of real devices. Emulation, on the other hand, executes the actual services and programs in a containerized environment. In other words, emulation can be used to achieve a higher level of realism. However, this adds a cost of extra computational complexity. Since the focus of this thesis is on studying the interactions between the attacker and defender agents, simulation provides enough realism and is thus preferred.

4.3 Attacker Agent Design

As previously mentioned, the RL-based attacker is tasked to classify nodes in a network as either real hosts or honeypots. However, initially the agent does not have any knowledge about the environment. This partially observable environment needs to be explored to gather information which can be used by the agent to classify the nodes in the graph. The agent can gather information about the network configuration, the devices present and their behavior. In a real-world scenario, an attacker can gather this information using various means such as, observing network traffic, analyzing device responses, detecting exposed endpoints, and identifying inconsistencies in the nodes' behavior [MC21].

The attacker's action space is designed to balance realism, expressive power, and computational feasibility. In practice, attackers do not immediately classify a system, but instead gather information in stages before making a decision. The actions listed in Table 1 reflect this staged reconnaissance process in an abstract but meaningful way.

ID	Action Type	Action	Description
0	Ping	Ping	A basic reachability and responsiveness test used to determine whether a device is present.
1	Probe	Syn-scan	A discovery action that tests whether network ports are open by observing how a device responds to connection initiation attempts, without fully establishing a connection.
2	Probe	Banner grab	A probing action that collects service-provided identification information by observing the initial message or metadata a device provides when a service is contacted.
3	Probe	OS-probe	A probing action that infers the operating system or platform of a device by analyzing characteristic response behavior.
4	Probe	Service probe	A probing action that evaluates the behavior and capabilities of an exposed service by interacting beyond initial identification to assess its depth, consistency, and realism.
5	Classify	Classify real	This action is used for the agent to classify a device as a legitimate host.
6	Classify	Classify honeypot	This action is used for the agent to classify a device as a honeypot.
7	No-op	No-op	This action entails doing nothing and waiting for the next timestep.

Table 1: Attacker agent action space with real-world descriptions

The probing actions (Ping, SYN scan, banner grabbing, OS probing, and service probing) represent increasingly informative steps in network reconnaissance. They differ in both cost and the amount of information they provide, allowing the agent to learn when lightweight exploration is sufficient and when deeper probing is worthwhile. This reflects realistic attacker behavior, where more aggressive probing can be informative but also riskier.

The two classification actions (classify real host and classify honeypot) explicitly separate information gathering from decision making. Classification is therefore a deliberate commitment rather than an automatic outcome of probing, allowing the agent to learn how much evidence is needed before labeling a host. The asymmetric penalties for misclassifications reflect the higher cost of false positives in realistic attack scenarios.

The no-op action enables the agent to model deliberate inaction. In a budget-constrained environment, this allows the attacker to avoid unnecessary probing when additional information is unlikely to improve the final decision.

The eight actions together form a minimal but complete abstraction of attacker behavior for this study. Reducing the action space would remove important distinctions between probing and classification. Expanding it with additional probe types or exploit-specific actions would mainly add complexity without providing new insights. The chosen action set therefore reflects a deliberate trade-off between realism and learning feasibility.

4.4 Defender Agent Design

The RL-based defender agent's goal is to make it as difficult as possible for an attacker to distinguish honeypots from real devices. Previous research in honeypot deception has already identified many techniques a RL-based defender can employ to achieve this goal [JJT+24]. Some of these techniques include:

- Imitating the imperfections and inconsistencies of real devices,
- Allowing honeypots to initiate connections, rather than only responding,
- Embedding honeypots within real devices to disguise their presence,
- Introducing stochastic jitter in the response times to simulate real-world latency variations.

The defender's action space is designed to capture the most important ways in which honeypots can be made harder to distinguish from real devices, while keeping the learning problem feasible. Rather than allowing arbitrary configuration changes, the defender is restricted to actions that directly affect properties commonly used by attackers during honeypot fingerprinting. Table 2 outlines the defender's actions and their effects on the network.

The selected actions focus on timing behavior, service identification, operating system consistency, artefact leakage, and exposed service surface. These properties are frequently used by attackers to identify honeypots and are therefore natural targets for defensive adaptation. By adjusting these parameters, the defender can influence exactly the signals the attacker observes during probing, without altering the underlying network structure.

Each action controls a distinct aspect of a host's observable behavior. Timing jitter affects timing-based detection, banner noise influences service fingerprinting, OS fingerprint adjustments target platform consistency, artefact probability models virtualization traces, and service count modifies perceived functionality. This separation ensures that the action space is expressive without being redundant.

The action space is deliberately limited to incremental modifications of individual host characteristics. More complex defensive strategies, such as dynamic topology changes, honeypot relocation, or traffic injection, are not included, as they would significantly increase the complexity of the environment and obscure the interaction between attacker probing strategies and defender deception. The chosen actions therefore represent a focused abstraction that preserves the essential dynamics of honeypot evasion while remaining suitable for reinforcement learning.

ID	Action	Description
0	Increasing RTT jitter	A defensive technique that increases variability in round-trip times to disrupt timing consistency and hinder timing-based fingerprinting.
1	Decreasing RTT jitter	A defensive technique that decreases variability in round-trip times to stabilize response times and hinder timing-based classification.
2	Increasing banner noise	A technique that increases ambiguity in service banners to reduce the reliability of banner-based fingerprinting.
3	Decreasing banner noise	A technique that decreases ambiguity in service banners to make the honeypot appear more realistic.
4	Increasing artefact probability	Increases the likelihood that a host exhibits artificial system artifacts indicating a synthetic deployment.
5	Decreasing artefact probability	Reduces the likelihood that a host exhibits artifacts that might suggest it is a honeypot.
6	Service count	Adjusts the number of visible network services on a device, influencing its perceived functionality.
7	OS fingerprint	Modifies the observed behavior of the host to more closely resemble a legitimate operating system.

Table 2: Defender agent action space with real-world descriptions

These actions are aimed at enhancing the realism of honeypots by making their behavior less predictable and more similar to that of real devices.

4.5 Adversarial Arena Design

Finally, the adversarial arena design corresponds to the third and final experiment, in which the attacker and defender agents will compete against each other. In the earlier experiments, each agent interacted with its own isolated environment. However, in the adversarial setting, the two agents will share the same environment. This causes the strategy of an agent to be influenced by the actions of the opponent.

In this setting:

- The attacker's action space will consist of actions that probe and classify devices, influencing the state of the environment.
- The defender's action space will focus on modifying the honeypots' characteristics, directly affecting the attacker's ability to identify them.

Additionally the defender no longer receives feedback based on conventional detection heuristics. Instead, feedback will be based on the success or failure of the attacker's actions. If the attacker successfully detects a honeypot, the defender will receive negative feedback, and vice versa.

5 Experimental Setup

The previous section described on a conceptual level how the environment and agents are designed and what choices and abstractions are made. This section will explain the technical setup for the experiments.

5.1 Honeypot Gym Environment Implementation

In this section, the implementation of the Honeypot Gym Environment, a core component of the experimental setup, is described. For most modern Reinforcement Learning (RL) applications, a simulated environment is essential for training and evaluating agents. The most commonly used framework for such simulations is the Gym environment, which provides a standardized interface for developing RL tasks. [TKT+24]

A Gym environment defines several key components that guide how agents interact with it. These components are essential for structuring the learning process and ensuring that the agent receives meaningful feedback. At the core of a Gym environment, the following components are defined [Wol25]:

- **Observation Space**: This represents all the information an agent receives about the current state of the environment. It includes data such as network configurations, the status of devices (real hosts or honeypots), and any other relevant variables the agent needs to make decisions.
- **Action Space**: The action space defines all the possible actions the agent can take at any given time. In the context of our experiment, these actions could involve probing devices, classifying hosts, or adjusting their configurations (in the case of the defender).
- **Reward Function**: The reward function provides numerical feedback to the agent based on the actions it performs. It tells the agent how well its actions contribute to achieving its objective. The reward function is designed to guide the agent toward its goal, whether that's detecting honeypots for the attacker or hiding them for the defender.
- **Transition Dynamics**: Transition dynamics describe how the environment changes in response to an agent's actions. These dynamics are usually unknown to the agent and must be learned through interaction. For instance, an attacker's action of scanning a host might reveal more information about the device's configuration, altering the state of the environment.
- **Episode Termination Conditions**: These conditions define when an episode should end. Termination can occur under various conditions, such as achieving a success criterion or hitting a time limit. These termination conditions help to structure the training process by providing clear endpoints for the agent to learn from.

Together, these components form the foundation of the Honeypot Gym Environment. They allow for the systematic training of the attacker and defender agents. The learning loop follows a cycle where the agent observes the current state, takes an action based on its policy, receives feedback in the form of rewards, and transitions to a new state. This process continues until the episode ends, either due to the completion of a task or reaching the time limit. [SB18]

5.1.1 Attacker Agent Implementation

In the following subsection the specific implementations of the previously mentioned components will be described for the attacker agent.

Observation Space The environment provides a strctured, graph based observation that represents the agent's current knowledge of the network, individual hosts, and remaining operational budget. The observation O_{att} is composed of 5 components:

$$O_{\text{att}} = \{X, A, m_{\text{known}}, m_{\text{classified}}, b_{\text{rem}}\} \tag{7}$$

where where each component has fixed dimensionality based on a maximum network size $N_{\max}$.

The first component, the node feature matrix X aggregates all probe outcomes and interaction history for that host. Each row corresponds to a host in the network and contains 34 different features. Table 3 gives an overview of the features per node.

Index	Description
0	Total number of probes performed
1	Number of observed open ports
2	Number of observed closed ports
3	Mean RTT (running mean)
4	RTT variance accumulator
5	Maximum banner score observed
6	Maximum OS fingerprint score observed
7	Honeypot artefact counter
8	Probe diversity (number of distinct probe types used)
9	Banner–OS inconsistency measure
10-29	Per-probe presence flags (historical encoding)
30	SYN scan performed
31	Banner grab performed
32	OS probe performed
33	Service probe performed

Table 3: Description of features per node of attacker observation space

This representation is history-aware but compact, enabling the policy to reason about uncertainty, probe sufficiency, and internal consistencies that may indicate honeypot behaviour. Explicit, probe-completion flags (feature 30-33) are included gate classification actions.

The next component, the discovered adjacency matrix A, represents the discovered network topology, not the true topology. An edge exists if and only if the agent has revealed the connection through probing. Separating discovered from true adjacency enforces partial observability and encourages strategic exploration of the network structure.

Next, the known host mask m_{known}, indicates whether the agent has discovered the existence of a host. Hosts become known via ping actions or by being revealed through neighboring discoveries. This mask is mainly used to mask the actions of unknown nodes to the agent.

The next component is the classified host mask $m_{\text{classified}}$. This component masks hosts that have already been classified as either real or honeypot. Classified hosts are terminal with respect to further interaction. The masking of actions for already classified hosts has been implemented to speed up the learning process by enforcing no time is being wasted on investigating nodes that have already been classified. This also further stimulates initial correct guessing as there is no possibility to change the classification retroactively.

Finally the last component, the remaining budget b_{rem}, provides the fraction of the amount of timestep still left. This fraction is described as:

$$b_{\text{rem}} = \frac{\text{remaining actions}}{\text{max actions}} \tag{8}$$

Normalizing the remaining budget enables time-aware decision making and encourages efficient early classification rather than exhaustive probing.

In addition to the core observation, the environment provides an action mask. This mask enforces environment constraints, such as: preventing classification before sufficient evidence, disabling actions on already classified hosts, limiting ping to once per host. Although not part of the physical state, the action mask defines the feasible action set at each timestep and is required for stable PPO training.

Action Space The attacker agent operates in a discrete, host-parameterized action space. Each action selects one host and one action type applicable to that host. Formally, the action space is defined as:

$$\mathcal{A} = \{0, 1, \ldots, N_{\text{max}} \cdot K - 1\} \tag{9}$$

where N_{max} is the maximum number of hosts in the network and $K = 8$ is the number of possible action types per host. Each action index $a \in \mathcal{A}$ is decoded into a host-action pair (h, k) as:

$$h = \left\lfloor \frac{a}{K} \right\rfloor \tag{10}$$

$$k = a \bmod K \tag{11}$$

where $h \in \{0, ..., N_{max} - 1\}$ denotes the target host, and $k \in \{0, ..., K - 1\}$ denotes the action type. As previously described the set of possible actions per host is defined in Table 1. Due to partial observability and operational constraints, not all actions are valid in every state. An action mask $m_a \in \{0, 1\}^{N_{max} \cdot K}$ is provided alongside the observation to indicate the set of legal actions. Invalid actions include, but are not limited to: interactions with non-existent hosts, actions on already classified hosts, classification actions before sufficient probing evidence is collected. The mask is enforced during policy sampling and prevents the agent from selecting illegal actions. This mask serves as initial guiding for the agent and helps speeding up the learning process.

Reward Function The reward function is designed to balance information gathering efficiency with accurate host classification, while reflecting the asymmetric costs associated with misclassification in realistic attacker scenarios.

At each timestep t, the agent receives a scalar reward $r_t \in \mathbb{R}$ composed of multiple additive components:

$$r_t = r_{step} + r_{probe} + r_{intrinsic} + r_{classification} + r_{termination} \tag{12}$$

Step cost incurs a small negative cost to discourage unnecessary interactions and promote efficient probing: $r_{\text{step}} = -0.01$.

To incentivize exploration, probe actions provide a positive reward depending on their informativeness:

$$r_{\text{probe}} = \begin{cases} 0.05, & \text{(SYN scan)} \\ 0.20, & \text{(Banner grab)} \\ 0.20, & \text{(OS probe)} \\ 0.25, & \text{(Service probe)} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

These rewards encourage higher-value probes while maintaining a trade-off with the action budget.

Certain probe responses may exhibit honeypot-specific artefacts. When such an artefact is observed, the agent receives a small intrinsic reward:

$$r_{\text{intrinisc}} = \begin{cases} 0.1, & \text{if a honeypot artefact is detected} \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

This signal is intentionally weak to avoid shortcut learning while still providing guidance during exploration.

Classification actions yield asymmetric rewards, reflecting the higher cost of false positives in security contexts.

$$r_{\text{classification}} = \begin{cases} +4, & \text{correct classification} \\ -6, & \text{real host classified as honeypot} \\ -3, & \text{honeypot classified as real} \end{cases} \tag{15}$$

This structure biases the agent toward caution when labeling real hosts as honeypots.

Some probes may actively trigger honeypot artefacts. When this occurs, an additional penalty is applied: $r_{\text{artefact}} = -3.0$. This penalty does not terminate the episode but discourages reckless interaction with deceptive services.

If the action budget is exhausted before all hosts are classified, a penalty proportional to the number of remaining unclassified hosts is applied:

$$r_{\text{budget}} = -2 \cdot U \tag{16}$$

where U is the number of unclassified hosts. Conversely, if the agent successfully classifies all hosts before the budget is exhausted, it receives an early termination bonus:

$$r_{\text{early}} = \frac{\text{remaining actions}}{\text{max actions}} \tag{17}$$

Transition Dynamics The environment dynamics are determined by a stochastic transition function that captures noisy probe outcomes, partial network discovery, and budget-constrained interactions. Given a state s_t and action a_t, the environment samples a successor state s_{t+1} according to:

$$s_{t+1} \sim P(s_{t+1}|s_t, a_t) \tag{18}$$

The transition function updates the agent's state along the following dimensions.

Network Discovery The true network topology is fixed for the duration of an episode but initially unknown to the agent. Discovery actions (e.g. ping) reveal neighboring hosts and edges in the discovered adjacency matrix. Newly revealed hosts are marked as known, while undiscovered hosts remain masked. The true topology itself is never revealed directly.

Probe Outcome Stochasticity All probe actions produce noisy observations. Transition dynamics include stochasticity in: round-trip time (RTT) measurements, banner and OS fingerprint scores, detection of honeypot-specific artefacts. This noise is sampled from host-specific response models and prevents deterministic inference from single observations.

Feature Updates Probe outcomes deterministically update the node feature matrix via aggregated statistics. These features provide a compact summary of the interaction history with each host.

Classification Transitions Classification actions transition a host into a terminal classified state. Once classified, a host can no longer be probed or reclassified. The assigned label is final and contributes to the terminal reward.

Budget Dynamics Each action reduces the remaining action budget:

$$b_{rem}^{t+1} = \frac{\text{remaining actions}_t - 1}{\text{max actions}} \tag{19}$$

No further actions are permitted when the budget is exhausted, resulting in episode truncation.

Episode Termination Conditions Episodes in the honeypot detection environment are finite and conclude under two distinct conditions: natural termination and budget-based truncation.

An episode terminates naturally when the agent has classified all hosts in the network. Formally, termination occurs when:

$$\sum_{h=0}^{N_{\text{actual}}-1} m_{\text{classified}}(h) = N_{\text{actual}} \tag{20}$$

At natural termination, no further actions are permitted. An early-completion bonus is applied based on the remaining action budget, rewarding efficient decision making.

If the maximum number of allowed actions is reached before all hosts are classified, the episode is truncated. Let $T_{\max}$ denote the action budget. Truncation occurs when: $t \geq T_{\max}$. At truncation, classification is incomplete. A penalty proportional to the number of remaining unclassified hosts is applied to the final reward.

When the agent selects an action that is invalid under the current action mask, the environment does not terminate the episode. Instead, a small penalty is applied and the state remains unchanged apart from the reduced remaining budget.

This design choice improves training stability and encourages the agent to respect action constraints without abruptly ending episodes.

5.1.2 Defender Agent Implementation

In the following subsection the specific implementations of the previously mentioned components will be described for the defender agent.

Observation Space The defender interacts with the same simulated enterprise network as the attacker, but receives a more compact, fully observed view of the current network state. The defender is assumed to operate as a privileged controller that already knows which devices are real and which are honeypots, and can directly observe their current "network signature" parameters.

At each timestep the defender receives an observation

$$O_{\text{def}} = \{X, A, b_{\text{rem}}\} \tag{21}$$

where $X \in \mathbb{R}^{N_{\max} \times F}$ is the node feature matrix, $A \in \{0,1\}^{N_{\max} \times N_{\max}}$ is the padded adjacency matrix of the underlying network, and $b_{\text{rem}} \in [0, 1]$ encodes the remaining interaction budget. The environment fixes a maximum network size $N_{\max}$. Episodes with fewer than $N_{\max}$ hosts are zero-padded in both X and A.

The node feature matrix X aggregates per-host metadata and response parameters into a fixed-length feature vector for each node. Table 4 gives an overview of the features per node.

Index	Description
0	Honeypot flag
1-4	Device type one-hot encoding
5	RTT mean
6	RTT jitter (RTT standard deviation)
7	Banner noise
8	OS fingerprint score
9	Artefact probability
10	Service count
11	Normalized degree

Table 4: Description of features per node of attacker observation space

The honeypot flag is a binary indicator specifying whether the host is a honeypot or a real device. This signal is only available to the defender and reflects the conceptual setting where the defender itself decides which nodes are deployed as honeypots. It allows the policy to learn how to adjust honeypot signatures, rather than where honeypots are placed.

Each host is assigned a device type from the set $\{\text{linux\_web}, \text{win\_desktop}, \text{iot\_sensor}, \text{db\_server}\}$. The device type one-hot encoding categorical variable enables the policy to condition its decisions on high-level device semantics (e.g. servers vs. endpoints) without having to infer them from low-level statistics.

The remaining entries capture the host's current network-level response characteristics, i.e. exactly the properties an attacker might use for honeypot fingerprinting. For each host, the following quantities are included and normalized. The RTT mean (average round-trip time) of responses is clipped to $[1, 200]$ ms and scaled to $[0, 1]$ by division by 200. The variability in round-trip times is clipped to $[0.5, 40]$ ms and scaled by division by 40. The banner noise is a value in $[0, 1]$ representing the amount of randomness or ambiguity in service banners and identification strings. The OS fingerprint score is a continuous similarity score between the host's observed behaviour and that of a genuine operating system, clipped to $[0.3, 1.5]$ and normalized by division by 1.5. Lower values correspond to synthetic signatures that are more typical of honeypots. The artefact probability is the probability that the host exhibits obvious virtualisation or deployment artefacts, in $[0, 1]$. The service count is the number of externally visible services, clipped to the range $[1, 10]$ and scaled by division by 10, representing the host's exposed functional surface. The normalized degree is the host's graph degree (number of neighbours) divided by $\max(1, N_{\text{actual}} - 1)$, capturing how centrally the host is embedded in the network topology.

Together, these features provide the defender with a snapshot of the current deception quality of each host: how realistic its timing behaviour is, how noisy or polished its banners are, how consistent its OS fingerprint appears, how many artefacts it leaks, how many services it exposes, and how well it is embedded in the network. This aligns with the conceptual design goals described in the defender design section, where the defender aims to make honeypots statistically indistinguishable from real devices by adjusting exactly these parameters.

The adjacency matrix A encodes the full, true network topology for the episode. Unlike the attacker's partially discovered adjacency, the defender receives the complete connectivity structure from the start of the episode. Each entry $A_{ij} = 1$ denotes a bidirectional link between hosts i and j, and entries outside the first N_{actual} rows and columns are zero-padded. This full-graph view allows the defender to reason about where honeypots are placed within the topology (e.g. core vs. edge nodes) and to adjust their signatures in a way that is consistent with their structural role.

Finally, the remaining budget b_{rem} (8) is a scalar in $[0, 1]$ that encodes how many steps the defender may still take in the current episode. This value is broadcast to all nodes and concatenated as an additional per-node feature in the implementation, enabling the policy to adapt its strategy based on how much time is left. Early in an episode the agent can afford larger, exploratory adjustments to host signatures, whereas near the end it is encouraged to make small, targeted refinements that directly reduce honeypot detectability without increasing the detectability of real hosts. Conceptually, the defender's observation is thus a fully observed, graph-structured state consisting of host metadata, network topology, and a normalized time/budget signal.

Action Space At each timestep the defender can modify the network signature of exactly one host and one feature. The environment exposes a discrete action space (9) where N_{max} is the maximum number of hosts in the network and $K = 8$ is the number of action types available per host. Each action index $a \in \mathcal{A}$ is decoded into a host-action pair (h (10), k (9)) so that the policy implicitly chooses which host to modify and which kind of tweak to apply. If $h \geq N_{\text{actual}}$ (i.e. an index that falls into the padded part of the observation), the action is ignored.

Each sub-action applies a small, bounded update to one of the parameters in the host's Response-Model (RTT statistics, banner noise, OS fingerprint, artefact probability, or service count). Most updates move the host towards a typical real-host baseline. One sub-action deliberately allows the defender to make a clearly bad change, so the RL agent must learn to avoid it. For every host i, the eight actions as described in Table 2 are available.

Each action applies a simple linear interpolation

$$x_{\text{new}} = (1 - \alpha)x_{\text{current}} + \alpha x_{\text{target}}, \tag{22}$$

for an appropriate parameter x (e.g. RTT standard deviation) and step size α that depends on whether the host is a honeypot or a real device and on its current value. After every update, all parameters are clamped to realistic bounds (e.g. $\text{rtt\_std} \in [0.5, 40]$, $\text{rtt\_mean} \in [1, 200]$, $\text{banner\_noise} \in [0, 1]$, $\text{service\_count} \in \{0, ..., 10\}$).

This design ensures that: the agent learns gradual, stable modifications instead of extreme jumps, all updated host profiles remain physically plausible, the defender can iteratively polish honeypot signatures over multiple timesteps, without changing the underlying network topology itself.

In summary, the discrete action space is structured as N_{max} blocks of 8 semantically meaningful actions. Each block corresponds to one host, and each sub-action to a specific, interpretable change of that host's network signature.

Reward Function The defender is trained to make honeypots harder to detect, while ensuring that real hosts do not become suspicious in the process. This is implemented via a shaped per-step reward that is entirely based on a heuristic detectability metric computed for each host.

Before defining the reward, the environment first constructs a per-episode baseline for what a typical real host looks like. At the beginning of each episode, it collects all REAL hosts and computes simple statistics over their response parameters and graph structure: RTT standard deviation, banner noise, OS fingerprint score, artefact probability, service count, node degree in the network graph.

For each of these features, the environment stores a mean and standard deviation. This defines the reference distribution for "normal" behaviour in the current episode. Given this baseline, a detectability score is computed for each host. For each host h, the score: computes z-scores for the main features (RTT std, banner noise, OS fingerprint, service count, graph degree) relative to the real-host baseline, clips each z-score into a bounded range, adds extra penalties for too low RTT jitter and high artefact probability, and combines everything into a weighted sum and clips the final score to $[0, 10]$. Higher scores mean more suspicious behaviour compared to typical real hosts.

Then these scores are aggregated into two averages: mean detectability over all honeypots, and mean detectability over all real hosts. At the start of each episode (before any defender action), these two values are stored as baselines and updated per step.

At each environment step, after the defender applies an action, the environment recomputes the mean detectability for honeypots and real hosts and then constructs a scalar reward. This rewards is computed according to the following formula:

$$r_t = \underbrace{(\text{prev\_h} - \text{mean\_h})}_{\text{improvement for honeypots}} - \underbrace{0.3 \cdot \max(0, \text{mean\_r} - \text{baseline\_detect\_real})}_{\text{penalty if real host becomes suspicious}} - 0.01. \tag{23}$$

This reward formula is composed of a few different terms. The honeypot improvement term $(\text{prev\_h} - \text{mean\_h})$ will give a positive reward, if the average detectability of honeypots decreases compared to the previous timestep. If honeypots become more detectable, this term is negative and the defender is penalised.

The real-host penalty term $(0.3 \cdot \max(0, \text{mean\_r} - \text{baseline\_detect\_real})$ averages the detectability of real hosts compared to its episode baseline. If real hosts remain at or below their baseline detectability, this penalty is zero. If real hosts become more suspicious than in the initial configuration, the defender is punished in proportion to how much worse they look. The coefficient 0.3 controls how strongly the reward reacts to this effect.

A small negative reward is applied at every timestep to slightly discourage unnecessarily long sequences of small, ineffective changes. In combination with the discount factor used by PPO, this encourages the policy to make efficient improvements early in the episode rather than procrastinating.

This reward design encodes three core objectives for the defender: making honeypots look less suspicious over time, do not sacrifice the realism of actual production hosts, and act efficiently

In summary, the reward function is a shaped signal that uses a simple heuristic attacker model (the detectability score) to approximate how well the defender is camouflaging honeypots, while explicitly constraining it not to degrade the apparent legitimacy of real hosts.

Transition Dynamics The defender environment implements deterministic transition dynamics conditioned on a fixed, but randomly initialised, network per episode. At the beginning of each episode, a new random graph is sampled, hosts are instantiated as either REAL or HONEYPOT with associated response parameters, and baseline statistics over REAL hosts are computed. Once this initial configuration is fixed, the evolution of the environment within an episode is fully determined by the defender's actions.

Given a state $s_t = \{X, A, b_{\text{rem}}\}$ and an action a_t, the environment applies the action to exactly one host and one parameter in its underlying response model. The discrete action index is decoded into a host index h and a sub-action k. If $h \geq N_{\text{actual}}$ (i.e. the action targets a padded node), the action is ignored and the observable state remains unchanged apart from the decreased budget. Otherwise, the corresponding host's response parameters are updated according to the chosen sub-action: RTT jitter, banner noise, artefact probability, service count, or OS fingerprint are nudged towards a target value using a simple linear interpolation step. Honeypots typically move more aggressively towards the real-host baseline than REAL hosts, reflecting the intuition that the defender should mainly "polish" honeypot signatures while only slightly adjusting real devices. After each update, all parameters are clamped to realistic ranges to ensure that host profiles remain plausible (e.g. RTT statistics, banner noise, artefact probability and service count all stay within predefined bounds).

The network topology itself does not change during an episode. The adjacency matrix A is fixed at reset time and only zero-padded for nodes beyond N_{actual}. As a result, the defender's actions only affect the labels and behavioural signatures of individual hosts, not their connectivity. However, the graph structure still influences the transition dynamics indirectly: node degree is part of the real-host baseline and of the detectability score, so changing a host's local parameters can have different effects depending on its topological position in the graph.

In addition to updating the host parameters, the environment maintains internal statistics that drive the reward function. After each action, a detectability score is recomputed for every host using the fixed REAL-host baseline. These scores are then aggregated into mean detectability values for honeypots and real hosts, which are compared against their previous values and episode baselines to construct the shaped reward signal. Although these quantities are not exposed directly in the observation space, they determine how the defender's choices influence future rewards and thus implicitly shape the optimal policy.

Finally, the remaining budget b_{rem} is updated deterministically at every step. The environment maintains a step counter and normalises the remaining number of steps by the fixed horizon $T_{\max}$, so that

$$b_{\text{rem}} = \frac{T_{\max} - (t+1)}{T_{\max}}. \tag{24}$$

An episode terminates when the maximum number of steps is reached. There is no separate success or failure condition in the defender baseline. This yields a finite-horizon Markov decision process in which the only stochasticity arises from the initial sampling of the network and host profiles at the beginning of each episode, while all within-episode transitions are deterministic given the current state and action.

Episode Termination Conditions Episodes in the defender environment have a fixed, step-based horizon and do not terminate early based on performance. The environment maintains an internal step counter $t \in \{0, ..., T_{\max}\}$, which is incremented by one after every call to step. An episode is marked as terminated as soon as the counter reaches the predefined maximum number of steps (denoted $T_{\max}$). Formally,

$$\text{terminated}_t = \begin{cases} \text{False}, & t < T_{\max}, \\ \text{True}, & t \geq T_{\max}. \end{cases} \tag{25}$$

The truncated flag is never used in the current implementation and is always set to False, so the only way for an episode to end is by exhausting the step budget.

There are no additional termination conditions based on reward thresholds, detectability metrics, or the state of the network. Even if honeypots become perfectly indistinguishable from real hosts, or conversely highly detectable, the episode continues until the maximum number of interaction steps has been reached. After termination, the training and evaluation scripts treat the episode as finished (done = terminated or truncated) and call reset to sample a new random network for the next episode.

5.1.3 Adversarial Arena Implementation

The third and final environment extends the previous single-agent setups into a shared, adversarial arena where the attacker and defender interact with the same underlying network. Concretely, the environment exposes a joint Gym interface with two agents: an attacker that probes and classifies hosts, and a defender that continuously mutates host response profiles to make honeypots harder to detect. Both agents act at every timestep, and their rewards are coupled in a near zero-sum fashion.

Formally, the environment maintains a single random network per episode (hosts, honeypot placements and true adjacency) and two different observation views on this shared state. The attacker's view is partially observed and derived from probe outcomes, whereas the defender receives a privileged, fully observed snapshot of the current host response parameters.

Joint Observation Space The adversarial arena is implemented as a dictionary-valued Gym environment

$$O = \{O_{\text{att}}, O_{\text{def}}\}, \tag{26}$$

where O_{att} and O_{def} are separate, structured observations for attacker and defender respectively.

The attacker observation O_{att} reuses the design of the single-agent honeypot detection environment and is defined as

$$O_{\text{att}} = \{X_{\text{att}}, A_{\text{disc}}, m_{\text{known}}, m_{\text{class}}, b_{\text{rem}}, m_{\text{act}}\}. \tag{27}$$

Here $X_{\text{att}} \in \mathbb{R}^{N_{\max} \times F_{\text{att}}}$ is the node feature matrix with $F_{\text{att}} = 34$ features per host, aggregating probe counts, running statistics over RTT, banner and OS scores, probe diversity and explicit probe-completion flags, exactly as described in Section 5.1.1. $A_{\text{disc}} \in \{0,1\}^{N_{\max} \times N_{\max}}$ is the discovered adjacency matrix, which is updated only when the attacker executes discovery actions (e.g. ping). It never reveals edges that have not been discovered. $m_{\text{known}} \in \{0,1\}^{N_{\max}}$ indicates which hosts are currently known to the attacker, either because they were pinged or discovered as neighbours. $m_{class} \in \{0,1\}^{N_{\max}}$ marks hosts that have already been classified as real or honeypot and are therefore terminal from the attacker's perspective. $b_{\text{rem}} \in [0,1]$ (8) encodes the shared action budget as a normalized scalar Finally, $m_{\text{act}} \in \{0,1\}^{N_{\max} \times K_{\text{att}}}$ is an action mask that encodes which attacker actions are currently legal. It disables actions on non-existent hosts, on already classified hosts, and prevents certain actions before sufficient evidence is collected (e.g. no classification if banner and OS probes have not yet been performed).

The defender observation O_{def} is a fully observed, compressed view of the same underlying network:

$$O_{\text{def}} = \{X_{\text{def}}, A_{\text{true}}, b_{\text{rem}}\}. \tag{28}$$

Here, $X_{\text{def}} \in \mathbb{R}^{N_{\max} \times F_{\text{def}}}$ with $F_{\text{def}} = 4 + 1 + 7$ encodes, per host: a honeypot flag (1 if honeypot, 0 if real), a one-hot device type indicator (linux web, Windows desktop, IoT sensor, database server), normalized response-model parameters: RTT mean and standard deviation, banner noise, OS fingerprint score, artefact probability, service count, and a normalized degree feature derived from the true graph. $A_{\text{true}} \in \{0,1\}^{N_{\max} \times N_{\max}}$ is the true adjacency matrix of the episode's network, padded to $N_{\max}$. Unlike the attacker, the defender always sees the full topology. Finally, the defender receives the same normalized remaining budget b_{rem} as the attacker, allowing both policies to condition their behaviour on how many attacker timesteps remain.

Both observations are static in dimensionality and padded up to the maximum network size $N_{\max}$ which allows the use of fixed-size Graph Neural Network policies for both agents.

Joint Action Space The environment maintains two separate discrete action spaces, one per agent: attacker actions:

$$\mathcal{A}_{\text{att}} = \{0, 1, ..., N_{\max} \cdot K_{\text{att}} - 1\}, \qquad K_{\text{att}} = 8, \tag{29}$$

and defender actions:

$$\mathcal{A}_{\text{def}} = \{0, 1, ..., N_{\max} \cdot K_{\text{def}} - 1\}, \qquad K_{\text{def}} = 8. \tag{30}$$

In both cases, an integer action a is decoded into a host–subaction pair (h (10), k (11)) where h indexes the host and k the action type for that host.

For each host, the attacker's action types are identical to those in Table 1: ping, four probe types (SYN scan, banner grab, OS probe, service probe), two classification actions (classify as real vs honeypot) and a no-op. The action mask m_{act} is recomputed at every step to enforce constraints such as: no actions on non-existent hosts, no further interaction with already classified hosts, no classification before both banner and OS probes have been performed for that host, and ping allowed at most once per host.

The defender's action space matches the eight network-signature modifications defined in Table 2. For a chosen host, the defender can increase or decrease RTT jitter, increase or decrease banner noise, adjust artefact probability, nudge the service count, or move the OS fingerprint towards more realistic values. If the selected host index falls into the padded region ($h \geq N_{\text{actual}}$), the action is ignored and the state remains unchanged except for the decreased budget.

Reward Coupling In the adversarial arena, the attacker's reward is computed locally based on its own probes and classifications, and the defender's reward is defined as the negative of the attacker's reward at each step. This yields a strictly competitive, near zero-sum game:

$$r_{\text{def}} = -r_{\text{att}} \tag{31}$$

The attacker reward function is a simplified version of the single-agent setup: Every action incurs a small step cost $r_{\text{step}} = -0.01$. Probe actions provide small positive rewards, with magnitude depending on their informativeness: SYN scan gives $+0.05$, plus an additional $+0.1$ if a honeypot artefact is observed, banner grab gives $+0.20$, plus $+0.1$ on artefact, OS probe gives $+0.20$, and service probe gives $+0.25$, plus $+0.1$ on artefact. Ping and no-op actions do not give intrinsic probe rewards (only the step cost). Classification actions use the asymmetric reward structure from the single-agent environment: $+4$ for correctly classifying a real host as real, $+4$ for correctly classifying a honeypot as honeypot, -6 for a false positive (real host classified as honeypot), -3 for a false negative (honeypot classified as real).

Illegal attacker actions ($m_{\text{act}}(a) = 0$) are penalised by a fixed negative reward and leave the state unchanged except for advancing the budget counter. Unlike the single-agent setting, there is no additional episode-level penalty for exhausting the budget, and no explicit early-completion bonus: the trade-off between probing and classifying is handled purely through the per-step rewards and the finite horizon.

Because the defender's per-step reward is defined as the negative of the attacker's, the defender is directly incentivised to push the network into configurations where the attacker earns low return: for instance by making honeypots hard to distinguish from real hosts so that misclassifications become more frequent or probes less informative.

Transition Dynamics and Step Ordering Each episode begins by sampling a new random network using the shared network generator. As in the previous environments, the generator samples: the number of hosts N_{actual}, which hosts are honeypots vs real devices, their device types and initial response-model parameters, and a connected random graph structure, padded to $N_{\max}$.

On top of this base state, the multi-agent environment maintains attacker-side state (probe histories, discovered adjacency, known/classified masks, budget), and defender-side state (mutable response models for each host).

At each timestep the environment first processes the defender agent's action and then attacker agent's action. The defender's action is decoded into $(h_{\text{def}}, k_{\text{def}})$. If $h_{\text{def}} < N_{\text{actual}}$, the corresponding host's response model is updated according to the selected sub-action: RTT jitter, banner noise, artefact probability, service count or OS fingerprint are nudged towards a target value and then clamped to realistic bounds. These changes immediately affect future probe outcomes for the attacker, but are not visible directly to the attacker. They can only be inferred through changes in observed RTT, banners and OS fingerprints.

The attacker's action is decoded into $(h_{\text{att}}, k_{\text{att}})$. If the action is legal under the current action mask, the environment executes the corresponding primitives. Ping actions reveal a host's existence, update RTT statistics and expose neighbours in the discovered adjacency. Probe actions sample noisy probe results from the host's current response model and update the attacker's node features (probe counts, maxima, variance accumulators, artefact counters, etc.). Classification actions mark a host as classified (with an immutable label) and compute the classification reward. No-op actions leave the environment unchanged except for the budget.The attacker's action counter is incremented every step, regardless of which action was taken.

From the perspective of each agent, the resulting process is still Markovian: the defender's next observation depends only on the current response models and topology, and the attacker's next observation depends only on the current probe features, discovered adjacency and classification/budget masks.

Episode Termination Episodes in the adversarial arena are finite and end under the attacker's constraints, because the attacker is the only agent that consumes the shared budget. If the attacker has classified all actual hosts in the network, i.e.

$$\sum_{h=0}^{N_{\text{actual}}-1} m_{\text{class}}(h) = N_{\text{actual}}, \tag{32}$$

the episode also terminates early.

There is no separate truncated signal in the current implementation Episodes end only via these two conditions. After termination, the environment returns final observations for both agents and the training loop resets the arena with a new random network.

In summary, the adversarial arena provides a compact yet expressive two-player RL testbed in which an RL-based attacker and defender co-evolve. The attacker operates under partial observability with a finite probe budget, while the defender continuously reshapes host signatures under full information. Their tightly coupled rewards turn honeypot detection and deployment into an explicit competitive game, allowing Experiment 3 to study emergent strategies and adaptation between the two agents.

5.2 Algorithms

Both the attacker and defender agents are trained using the Proximal Policy Optimization (PPO) algorithm, an on-policy actor–critic method introduced by Schulman et al. [SWD+17] that combines clipped policy updates with value function regression and an entropy bonus. PPO is well-suited to the honeypot setting because it supports discrete, high-dimensional action spaces, is robust to stochastic rewards, and provides more stable training compared to earlier policy gradient methods. While PPO does not explicitly address partial observability, it can be effectively applied in partially observable environments when combined with history-aware observations, as is the case in this work.

In addition, both policies are implemented as graph neural network (GNN) architectures that operate directly on the graph-structured observations produced by the environments. Rather than treating each host in isolation, the policy reasons over the full network topology and per-host features, allowing it to capture relational patterns such as centrality, neighbourhood structure, and consistency of host signatures across the graph. [Z+20]

5.2.1 Proximal Policy Optimization

Both agents are trained with the clipped Proximal Policy Optimization (PPO) algorithm [SWD+17]. PPO maintains two function approximators: a stochastic policy $\pi_\theta(a \mid s)$ (the "actor") and a value function $V_\phi(s)$ (the "critic"), parameterised by neural networks with parameters θ and ϕ respectively. In this work, the actor and critic share a GNN backbone and are optimised jointly.

Training proceeds in repeated cycles of on-policy data collection and policy updates. For each PPO update, the agent interacts with the environment for a fixed number of timesteps T, producing a trajectory of states, actions, rewards and value estimates:

$$\{(s_t, a_t, r_t, V_t)\}_{t=0}^{T-1}, \quad V_t \approx V_\phi(s_t). \tag{33}$$

From this trajectory, Generalised Advantage Estimation (GAE) is used to compute an advantage estimate $\hat{A}_t$ for each timestep:

$$\delta_t = r_t + \gamma V_{t+1} - V_t, \tag{34}$$

$$\hat{A}_t = \delta_t + \gamma\lambda \hat{A}_{t+1}, \tag{35}$$

where γ is the discount factor and λ is the GAE parameter. The target return for the value function is then

$$\hat{R}_t = \hat{A}_t + V_t. \tag{36}$$

To update the policy, PPO maximises a clipped surrogate objective that limits the size of each policy update:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\big(r_t(\theta)\,\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_t\big)\Big], \tag{37}$$

where the probability ratio is

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{38}$$

This objective prevents very large policy updates by clipping the ratio $r_t(\theta)$ to the interval $[1-\epsilon, 1+\epsilon]$ when it would otherwise increase the magnitude of the advantage. The clipping parameter ϵ is set to 0.15 in all experiments.

The full optimisation objective combines the clipped policy loss, a squared-error value loss and an entropy bonus to encourage exploration:

$$L(\theta, \phi) = \mathbb{E}_t \Big[L^{\mathrm{CLIP}}(\theta) - c_V \, (V_\phi(s_t) - \hat{R}_t)^2 + c_H \, \mathcal{H}[\pi_\theta(\cdot \mid s_t)] \Big], \tag{39}$$

where c_V controls the weight of the value loss and c_H the strength of the entropy regularisation. In the attacker experiments the discount factor is set to $\gamma = 0.98$ and $\lambda = 0.92$, while in the defender and multi-agent experiments a slightly more conservative choice $\gamma = 0.99$, $\lambda = 0.95$ is used to stabilise training in longer episodes.

Both environments provide an action mask that encodes the set of legal actions per timestep (e.g. preventing interaction with non-existent or already classified hosts). During training and evaluation, this mask is enforced by setting the logits of illegal actions to a large negative value, effectively removing them from the support of the policy distribution. This greatly improves training stability in the large $N_{\max} \cdot K$ action space.

Algorithm 1 summarises the PPO training loop as implemented for both agents.

Algorithm 1 PPO training for honeypot agents

```
1: Initialise policy parameters $\theta$ and value parameters $\phi$
2: Set global timestep counter $n \leftarrow 0$
3: **while** $n < N_{\text{total}}$ **do**                                   ▷ Collect on-policy rollout
4:     **for** $t = 0$ to $T - 1$ **do**
5:         Observe state $s_t$ and action mask $m_t$
6:         Sample action $a_t \sim \pi_\theta(\cdot \mid s_t, m_t)$
7:         Execute $a_t$ in the environment
8:         Receive reward $r_t$ and next state $s_{t+1}$
9:         Store $(s_t, a_t, r_t, m_t)$ in buffer
10:        $n \leftarrow n + 1$
11:        **if** episode terminated or truncated **then**
12:            Reset environment and continue
13:        **end if**
14:    **end for**                                   ▷ Compute advantages and returns (GAE)
15:    Compute value estimates $V_t = V_\phi(s_t)$ for all stored states
16:    Compute $\hat{A}_t$ and $\hat{R}_t$ using GAE and bootstrapped $V_T$
17:    Normalise advantages: $\hat{A}_t \leftarrow (\hat{A}_t - \mu_A)/(\sigma_A + 10^{-8})$      ▷ Multiple epochs of minibatch
   optimisation
18:    **for** epoch $= 1$ to $K_{\text{epochs}}$ **do**
19:        Shuffle rollout indices and split into minibatches $\mathcal{B}$
20:        **for** each minibatch $B \in \mathcal{B}$ **do**
21:            Compute policy logits with action masking for states in $B$
22:            Evaluate new log-probabilities $\log \pi_\theta(a_t \mid s_t)$ for $t \in B$
23:            Compute probability ratios $r_t(\theta)$ and $L^{\text{CLIP}}$
24:            Compute value loss and entropy bonus
25:            Take one gradient step on $L(\theta, \phi)$ with Adam
26:            Clip gradients to $\ell_2$ norm 0.5
27:        **end for**
28:    **end for**
29:    Discard buffer and start new rollout
30: **end while**
```

5.2.2 Graph Neural Network

Both the attacker and defender policies are implemented as graph neural networks that operate directly on the graph-structured observations described in Section 5.1. Instead of learning a separate embedding for each host in isolation, the network aggregates information over the full adjacency matrix and per-node feature vectors, allowing the policy to condition each decision on the local neighbourhood and broader network context. [Z+20]

Input representation. For the attacker, the environment constructs a node feature matrix $X \in \mathbb{R}^{N_{\max} \times F_{\text{att}}}$ that encodes aggregated probe statistics for each host (Table 3), together with a discovered adjacency matrix $A^{\text{disc}} \in \{0,1\}^{N_{\max} \times N_{\max}}$, a known-host mask and a classified-host mask. The PPO agent internally reconstructs these tensors from the flattened observation and concatenates the masks and normalised remaining budget as additional per-node features.

For the defender, the environment provides a node feature matrix $X \in \mathbb{R}^{N_{\max} \times F_{\text{def}}}$ that includes an explicit honeypot flag, a device-type one-hot encoding and normalised response parameters (e.g. RTT statistics, banner noise, OS fingerprint score, artefact probability, service count, degree), together with the full adjacency matrix $A \in \{0,1\}^{N_{\max} \times N_{\max}}$ and a scalar remaining-budget signal. The remaining budget is broadcast to all nodes and concatenated as an extra feature.

Graph convolution layers Both agents use the same simple message-passing layer, denoted GraphLayer, which can be viewed as a normalised graph convolution with separate linear transformations for self-features and neighbour features. Given node embeddings $H \in \mathbb{R}^{N \times d}$ and an adjacency matrix $A \in \{0,1\}^{N \times N}$, the layer first adds self-loops,

$$\tilde{A} = A + I_N, \tag{40}$$

and row-normalises:

$$\hat{A}_{ij} = \frac{\tilde{A}_{ij}}{\sum_k \tilde{A}_{ik} + 10^{-6}}. \tag{41}$$

Neighbour information is then aggregated as

$$H^{\text{neigh}} = \hat{A} H, \tag{42}$$

and combined with the original node embeddings through two learned linear projections:

$$H' = \text{LayerNorm}\big(\text{ReLU}(W_{\text{self}} H + W_{\text{neigh}} H^{\text{neigh}})\big), \tag{43}$$

where $W_{\text{self}}, W_{\text{neigh}} \in \mathbb{R}^{d \times d}$ are trainable weight matrices and LayerNorm is applied per node. Stacking two such layers allows the policy to aggregate information from nodes up to two hops away in the network.

Policy and value heads After two graph layers, each node i has a d-dimensional embedding $h_i \in \mathbb{R}^d$. The discrete action space is structured as $N_{\max}$ blocks of K actions, where $K = 8$ is the number of action types per host. The policy head is implemented as a shared linear projection

$$\ell_i = W_\pi h_i + b_\pi \in \mathbb{R}^K, \tag{44}$$

applied independently to each node. The per-node logits are then reshaped into a single vector of size $N_{\max} \cdot K$ to form the logits of the global categorical policy $\pi_\theta(a \mid s)$. Before constructing the categorical distribution, the environment-provided action mask is applied by setting the logits of invalid actions to a large negative constant, ensuring that illegal host–action combinations are never sampled.

For the value function, node embeddings are first aggregated by a simple mean pooling:

$$\bar{h} = \frac{1}{N} \sum_{i=1}^{N} h_i, \tag{45}$$

followed by a linear projection to a scalar:

$$V_\phi(s) = w_V^\top \bar{h} + b_V. \tag{46}$$

This global value estimate is used both for the critic loss and for GAE.

Parameter sharing across settings. The same graph-based architecture is used for the single-agent attacker, the single-agent defender, and the attacker/defender in the adversarial arena. Only the input dimensionality (attacker vs. defender feature vectors) and the observation flattening logic differ. The graph layers and action/value heads remain identical. This design choice ensures that any performance differences between experiments are due to changes in the environment and training setup, rather than architectural confounds, and highlights the generality of the GNN-based PPO approach for graph-structured honeypot environments.

5.3 Evaluation Metrics

This section defines the evaluation metrics used to quantify performance in the three experimental settings: the single-agent attacker, the single-agent defender, and the adversarial arena. Across all experiments, results are reported as averages over a fixed number of evaluation episodes, where each episode samples a new random network with variable size $N_{\text{actual}} \le N_{\max}$. Metrics that depend on classification outcomes are aggregated over all classification decisions across episodes, ensuring comparability despite varying episode sizes.

In addition to task-specific metrics, the average episodic return $\mathbb{E}\left[\sum_t r_t\right]$ is also reported, since all agents are trained with PPO to optimise this objective. However, because shaped rewards can obscure failure modes (e.g. trading off errors against efficiency), returns are complemented with explicit, interpretable performance measures.

5.3.1 Attacker Agent

The attacker's objective is to correctly classify every host as either REAL or HONEYPOT under partial observability and a finite probing budget. The environment terminates early when all hosts are classified, or ends via budget exhaustion when the maximum number of actions T_{max} is reached. In the latter case, a penalty proportional to the number of remaining unclassified hosts is applied.

Classification Metrics Whenever the attacker executes a classification action (classify REAL or classify HONEYPOT), the predicted label is compared to the true host type and accumulate a confusion matrix with: TP being correctly classified honeypots, TN correctly classified real nodes, FP falsely classified honeypots, and FN falsely classified real nodes. From these counts, the following is computed:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{47}$$

Coverage and Efficiency Since a strict action budget is operated under by the attacker, the efficiency with which complete classification is achieved is additionally measured through classification coverage ($\frac{\#\text{classified}}{N_{\text{actual}}}$), budget usage (the number of actions used until termination, normalised by T_{max}), and early termination frequency (fraction of episodes that end because all hosts are classified rather than because the budget is exhausted).

In this implementation, classification actions are only enabled once sufficient probe evidence has been collected (banner and OS probe completion flags). This gating makes the "efficiency" metrics meaningful: finishing early indicates not only fast classification, but also the ability to select informative probes and stop probing when evidence is sufficient.

5.3.2 Defender Agent

The defender's objective is to iteratively modify host response signatures such that honeypots become less detectable, while real hosts do not become more suspicious. The defender is trained and evaluated using a shaped reward derived from a heuristic detectability score computed relative to a per-episode baseline estimated from REAL hosts.

Detectability-Bases Metrics [SPV23] For each host h, the environment computes a detectability score $D(h) \in [0, 10]$ by combining clipped z-scores of key signature features (e.g. RTT jitter, banner noise, OS fingerprint, service count, graph degree), with additional penalties for unnaturally low jitter and high artefact probability. Higher values indicate stronger deviation from "typical real-host" behaviour.

These per-host scores are aggregated into:

$$\bar{D}_{\text{honeypot}} = \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} D(h), \tag{48}$$

$$\bar{D}_{\text{real}} = \frac{1}{|\mathcal{R}|} \sum_{h \in \mathcal{R}} D(h), \tag{49}$$

where $\mathcal{H}$ and $\mathcal{R}$ denote honeypot and real host sets respectively.

During evaluation, per episode, the mean of these quantities over timesteps is reported (since the defender acts multiple times per episode). The mean honeypot detectability averages $\bar{D}_{\text{honeypot}}$ across steps (lower is better), and the mean real-host detectability averages $\bar{D}_{\text{real}}$ across steps (should not increase).

Distributional Indistinguishability (KL) [ZLC+21] To quantify how similar honeypots become to real hosts at the feature-distribution level, the KL-divergence between REAL and HONEYPOT distributions over core response parameters (RTT mean/std, banner noise, OS fingerprint, artefact probability, and graph degree) is computed. In the implementation, each feature distribution is approximated as a 1D Gaussian and KL is averaged across features. Lower values indicate that honeypot signatures have become statistically closer to real hosts.

Heuristic Attacker Success Metrics Finally, the defender is evaluated against a fixed heuristic attacker that classifies hosts using the detectability score threshold τ:

$$\hat{y}(h) = \begin{cases} \text{HONEYPOT} & \text{if } D(h) \geq \tau, \\ \text{REAL} & \text{otherwise.} \end{cases} \tag{50}$$

From this rule the Attack Success Rate (ASR) and Honeypot Detection Rate (HDR) are computed. The ASR is the fraction of all hosts correctly classified by the heuristic attacker, and the HDR is the recall on honeypots, i.e. $\frac{TP}{TP+FN}$. In this setting, strong defender performance corresponds to lower ASR and lower HDR, since honeypots should become harder to identify while real hosts remain benign.

5.3.3 Adversarial Arena

In the adversarial arena, both agents interact with the same underlying network: the defender continuously mutates host signatures under full information, while the attacker probes and classifies under partial observability and a shared finite budget. Episodes terminate when the attacker exhausts its budget or when all hosts are classified.

Because rewards are coupled in a near zero-sum fashion (the attacker is harmed when the defender is rewarded), both sides of the interaction must be captured in the evaluation rather than only single-agent returns. Concretely, the following are reported: Average attacker return and average defender return over episodes, attacker classification metrics (Accuracy) are computed exactly as in Section 5.3.1, but now under defender interference, and classification coverage and budget usage, indicating whether longer probe sequences are forced by the defender or the likelihood of budget exhaustion is increased.

This combination of metrics allows for the distinction between (i) defenders that are able to truly reduce honeypot detectability and cause systematic attacker errors, versus (ii) defenders that mainly delay the attacker without having its final classification quality changed.

6 Experimental Results

6.1 Attacker Agent Performance

This subsection evaluates the attacker agent in isolation. The attacker is tasked with classifying network hosts as either *REAL* or *HONEYPOT* under partial observability and a finite probing budget.

Figure 1 illustrates the training progression of the attacker agent. Episodic return (left y-axis) and classification accuracy (right y-axis) are shown as functions of the training episode. Both metrics exhibit steady improvement during training and converge to stable values toward the end of training, indicating that the reward structure is well aligned with the classification objective and that the learned policy has converged prior to evaluation.

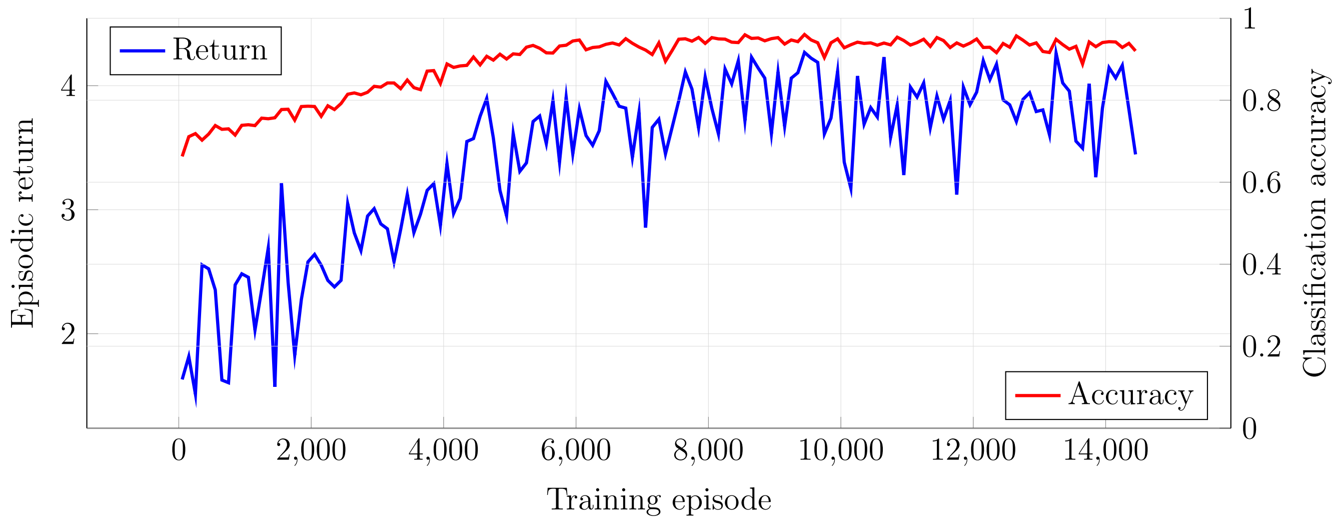

Figure 1: Training progression of the attacker agent showing episodic return (left y-axis) and classification accuracy (right y-axis) over training episodes.

Next, performance is evaluated over 1000 independent evaluation rollouts. No learning or policy updates occur during this phase. The primary metrics are classification accuracy and episodic return. An aggregate overview of the attacker's performance is provided in Table 5, while per-episode behavior is illustrated in Figures 2 and 3.

Across the full evaluation set, the attacker performs a total of 22104 host classifications. Of these, 20257 classifications are correct and 1847 are incorrect, resulting in an overall classification accuracy of 91.64%. The attacker achieves a mean episodic return of 84.96, with returns ranging from 6.67 to 183.10. These results indicate that correct classifications generally outweigh probing costs and misclassification penalties.

Metric	**Value**
Evaluation rollouts	1000
Total classifications	22104
Correct classifications	20257
Incorrect classifications	1847
Overall accuracy	91.64%
Mean episode return	84.96
Standard deviation of return	41.91
Minimum episode return	6.67
Maximum episode return	183.10
Mean classifications per episode	22.10
Mean incorrect classifications per episode	1.85

Table 5: Aggregate attacker performance over 1000 evaluation rollouts

As shown in Table 5, the attacker consistently classifies a substantial number of hosts per episode while maintaining a low average number of incorrect classifications. The relatively high standard deviation in episodic return reflects variability in episode length and host composition rather than instability in the learned policy.

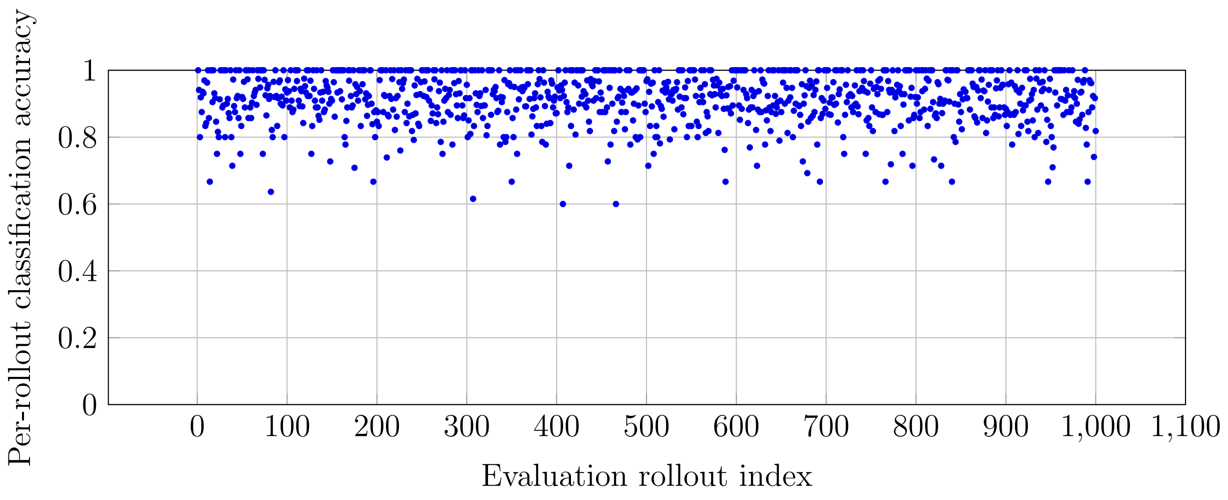

Figure 2: Per-rollout classification accuracy of the attacker agent over 1000 independent evaluation rollouts using a fixed trained policy. Each point corresponds to one evaluation episode on a newly sampled network instance. The ordering of rollouts on the x-axis is arbitrary and does not represent training progression.

Figure 2 shows that classification accuracy remains consistently high across evaluation rollouts.

This indicates that the trained policy generalizes well to previously unseen network configurations.

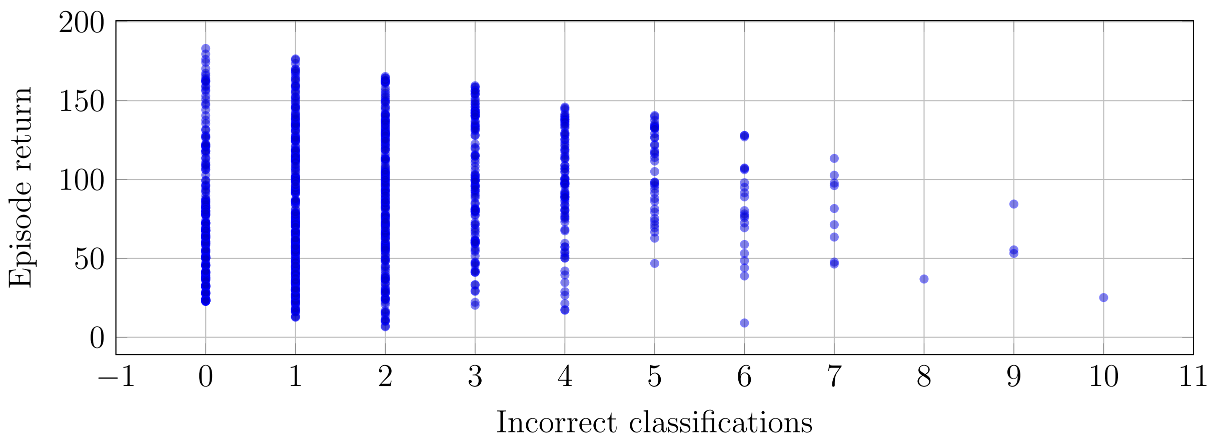

Figure 3: Relationship between episodic return and the number of incorrect classifications across evaluation rollouts. Episodes with more misclassifications yield lower returns, reflecting the asymmetric penalty structure of the attacker reward function.

Figure 3 illustrates the negative correlation between the number of incorrect classifications and episodic return. Episodes with a higher number of errors tend to produce lower returns, confirming that misclassifications are effectively discouraged by the reward structure.

Overall, the attacker agent demonstrates robust and stable performance in the single-agent evaluation setting. With an overall accuracy exceeding 91% and predominantly positive episodic returns, the attacker provides a strong baseline for subsequent experiments in which it operates against an adaptive defender.

6.2 Defender Agent Baseline

This experiment evaluates the defender agent in isolation. The defender's objective is to reduce the detectability of honeypots under conventional heuristic-based detection methods, while maintaining realistic behavior for real hosts. This experiment establishes a non-adversarial baseline prior to the adversarial arena evaluation.

Two defender strategies are compared: a Reinforcement Learning defender trained using Proximal Policy Optimization (PPO), and a random baseline defender that applies defensive actions uniformly at random.

Figure 4 illustrates the training progression of the PPO-based defender agent. The figure reports episodic return (left y-axis) and honeypot detection rate (HDR) (right y-axis) as a function of the training episode. During the early stages of training, episodic returns are lower and more variable, reflecting exploratory behavior and ineffective honeypot configuration strategies. As training progresses, the defender slowly improves its performance, achieving higher returns as it learns to deploy honeypots in a manner that reduces heuristic-based detectability while preserving realistic behavior for real hosts.

When examining the honeypot detection rate (HDR), a pronounced initial decline can be observed during the early training episodes. This sharp decrease indicates that the defender rapidly learns to manipulate host characteristics in ways that significantly reduce heuristic-based honeypot detection. As training continues, the rate of improvement diminishes and the HDR gradually plateaus, stabilizing at a value slightly below 0.2. The observed plateau suggests that the agent converges to a strategy that achieves a favorable trade-off between minimizing honeypot detectability and maintaining realistic behavior across the network.

The policy obtained at the end of training is used for all defender evaluation experiments reported in this section.

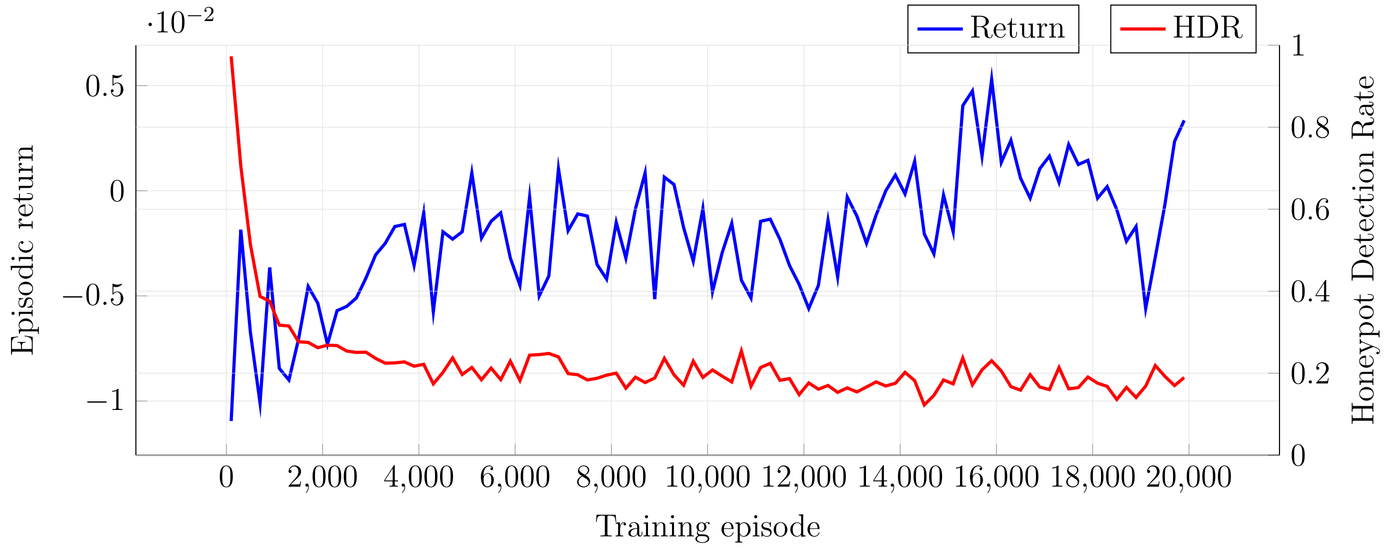

Figure 4: Training progression of the defender agent showing episodic return (left y-axis) and honeypot detection rate (right y-axis) over training episodes.

Both strategies are evaluated over 1000 independent evaluation rollouts, with no learning or policy updates during evaluation. Performance is measured using episodic return, Attacker Success Rate (ASR), Honeypot Detection Rate (HDR), mean detectability scores for honeypots and real hosts, and the KL divergence between honeypot and real host behavior distributions. An aggregate comparison is shown in Table 6, while per-rollout evaluation outcomes are illustrated in Figures 5 and 6.

Metric	**PPO Defender**	**Random Defender**
Evaluation rollouts	1000	1000
Mean episode return	5.31	-0.75
Mean Attacker Success Rate (ASR)	0.48	0.73
Mean Honeypot Detection Rate (HDR)	0.18	1.00
Mean honeypot detectability	3.89	8.03
Mean real host detectability	2.88	2.90
Mean KL-divergence	11.28	5.93

Table 6: Aggregate defender performance over 1000 evaluation rollouts

Table 6 shows that the PPO-trained defender substantially outperforms the random baseline across most evaluation metrics. The PPO defender achieves a positive mean episodic return of 5.31, whereas the random defender yields a negative mean return of -0.75, indicating ineffective honeypot deployment. The mean attacker success rate is reduced from 0.73 under random deployment to 0.48 when using the PPO defender. Similarly, the honeypot detection rate drops sharply from near-certain detection (HDR = 1.00) for the random baseline to 0.18 for the PPO defender, indicating that honeypots deployed by the PPO defender frequently evade heuristic-based detection.

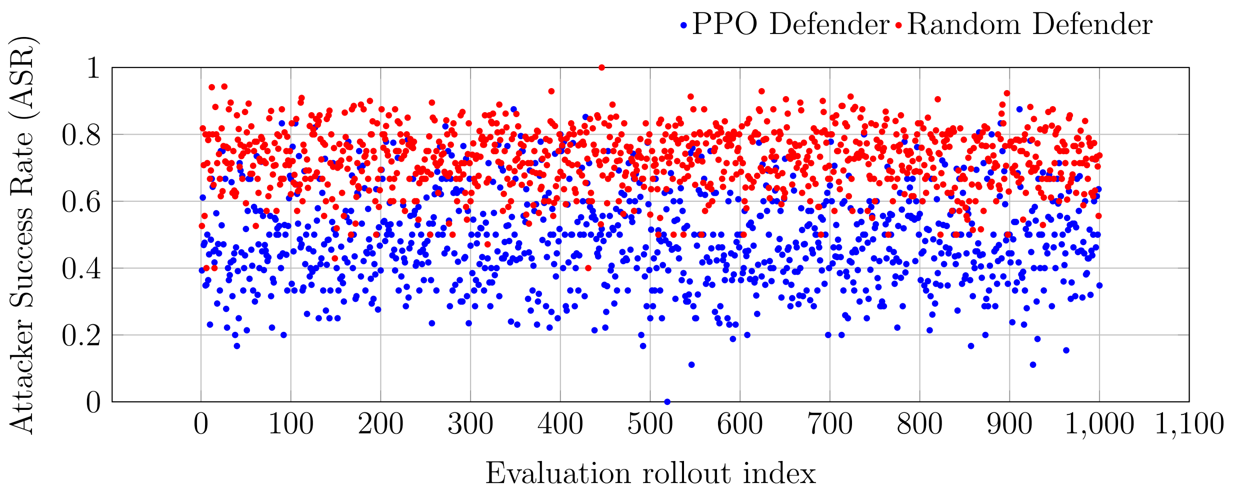

Figure 5: Per-episode attacker success rate for the PPO defender and random baseline across 1000 evaluation rollouts.

Figure 5 shows the distribution of attacker success rates across evaluation rollouts. For nearly all episodes, the PPO defender yields substantially lower attacker success rates than the random baseline, demonstrating the effectiveness of learned defensive strategies under heuristic attack.

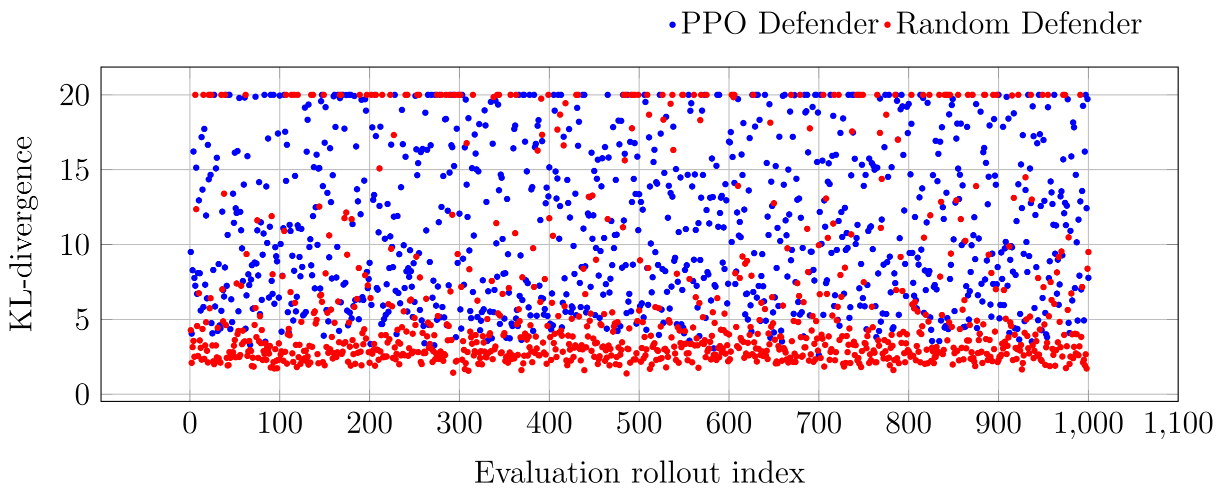

Figure 6: Per-rollout KL-divergence between honeypot and real host behavior distributions for PPO and random defenders.

Figure 6 reports the per-episode KL-divergence between honeypot and real-host feature distributions. Lower KL-divergence values indicate greater statistical similarity between honeypots and real hosts.

In contrast to the detectability-based metrics, the PPO defender does not achieve lower KL-divergence than the random baseline. On average, the PPO defender exhibits a higher KL-divergence (11.28) compared to the random defender (5.93). This indicates that while the PPO defender is highly effective at reducing heuristic detectability, it does not explicitly minimise full distributional similarity between honeypots and real hosts.

This discrepancy is expected given the reward design. The defender is trained to reduce a heuristic detectability score rather than to optimise KL-divergence directly. As a result, the learned policy focuses on suppressing the specific fingerprinting features used by heuristic detection methods, even if residual distributional differences remain across the full feature space.

These results highlight that low attacker success rates do not necessarily imply full statistical indistinguishability, and that heuristic robustness and distributional matching capture complementary aspects of honeypot deception quality.

Overall, the results demonstrate that Reinforcement Learning enables the defender to learn effective honeypot deployment strategies that significantly reduce detectability under heuristic-based analysis. Although honeypot detection is not eliminated entirely, the PPO defender achieves substantial improvements over random deployment across most evaluation metrics. These results justify the use of the PPO defender as a baseline for the adversarial arena experiment presented in the next section.

6.3 Adversarial Arena Results

This section evaluates the interaction between the RL-based attacker and RL-based defender in the adversarial arena using fixed, fully trained policies. In this setting, both agents act on the same network environment and pursue opposing objectives. The attacker aims to classify hosts as either *REAL* or *HONEYPOT* under partial observability and a finite probing budget, while the defender actively modifies host signatures to reduce detectability. As in the single-agent attacker experiment, classification accuracy is the primary performance metric, as it directly reflects which agent dominates the interaction in terms of task outcome.

Evaluation is performed over 1000 independent evaluation rollouts. Each episode is initialized with a newly sampled random network configuration and terminates when the attacker exhausts its probing budget or reaches a terminal classification state. No learning or policy updates occur during evaluation. All reported metrics are computed from environment ground-truth classification outcomes.

Classification Accuracy Table 7 summarizes the attacker's classification performance in the adversarial arena. Across all rollouts, the attacker performs a total of 17818 classifications, achieving a mean rollout accuracy of 0.77. This represents a substantial degradation compared to the isolated attacker setting, where accuracy exceeded 0.91, demonstrating that the defender is able to meaningfully interfere with the attacker's decision-making process.

Metric	**Value**
Evaluation rollouts	1000
Total classifications	17818
Mean classifications per rollout	17.82
Mean rollout accuracy	0.77
Minimum rollout accuracy	0.33
Maximum rollout accuracy	1.00

Table 7: Aggregate attacker classification performance in the adversarial arena.

Figure 7 shows the per-episode classification accuracy across evaluation rollouts. Accuracy varies substantially between episodes, ranging from near-perfect performance to episodes with pronounced misclassification. This variability reflects the stochastic and adversarial nature of the environment, in which defender actions dynamically alter host signatures and disrupt the attacker's learned probing heuristics. The ordering of episodes on the x-axis is arbitrary and does not represent training progression.

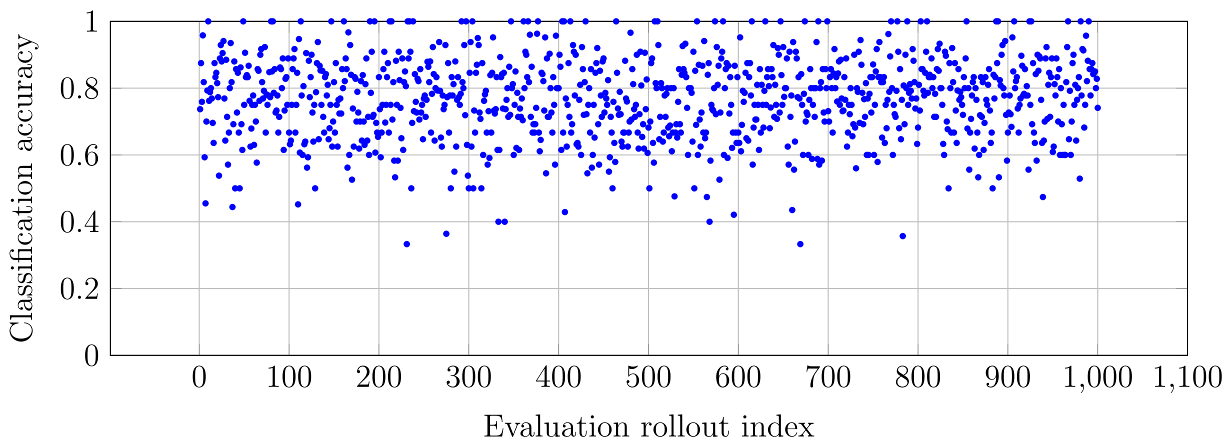

Figure 7: Per-rollout attacker classification accuracy in the adversarial arena.

Classification Errors To further analyze the source of accuracy degradation, Table 8 reports the mean number of correct and incorrect classifications per rollout. On average, the attacker makes approximately 4 incorrect classifications per rollout, indicating that defender-induced ambiguity does not merely delay decisions, but actively causes systematic misclassification.

Metric (per rollout)	**Mean**
Classified hosts	17.82
Correct classifications	13.72
Incorrect classifications	4.10

Table 8: Mean per-rollout classification outcomes in the adversarial arena.

Episodic Returns Although episodic return is not the primary evaluation metric, it provides insight into the efficiency and cost structure of the interaction. Table 9 reports attacker and defender returns. As expected from the zero-sum reward formulation, returns are strictly anti-correlated.

Metric	**Attacker**	**Defender**
Mean episode return	73.73	-73.73
Standard deviation	25.35	25.35
Minimum return	0.74	-145.75
Maximum return	145.75	0.74

Table 9: Episodic returns in the adversarial arena.

Figure 8 visualizes the distribution of attacker returns. The distribution is shifted towards lower values compared to the isolated attacker baseline, reflecting increased probing costs and misclassification penalties induced by the defender's actions.

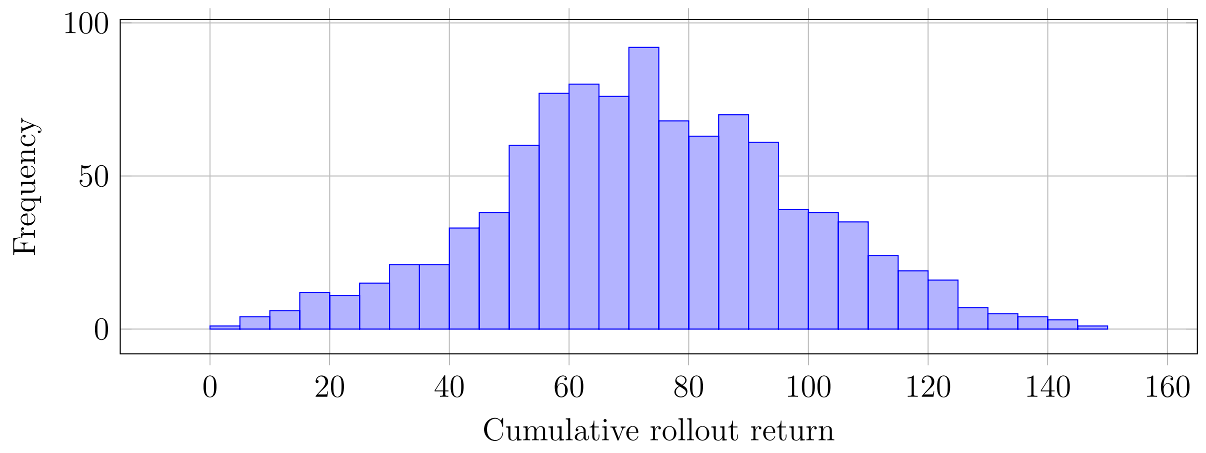

Figure 8: Histogram of attacker cumulative rollout returns in the adversarial arena.

Summary. The adversarial arena results reveal a clear asymmetry between the two agents. While the defender substantially degrades attacker performance relative to the single-agent baseline, the attacker retains outcome dominance with respect to its primary objective. A mean classification accuracy of approximately 75% indicates that the attacker continues to identify honeypots reliably, performing far above random chance even under active defensive interference.

At the same time, the reduction from over 90% accuracy in the isolated setting to approximately 75% in the adversarial arena demonstrates that the defender meaningfully disrupts the attacker's decision process. Rather than neutralising the attacker entirely, the defender increases ambiguity, variance, and misclassification frequency, thereby raising the cost and uncertainty of successful detection.

These results suggest that the adversarial interaction does not converge to a balanced equilibrium, but instead to a regime in which the attacker retains outcome dominance while the defender exerts significant resistance. From a defensive perspective, this implies that adaptive honeypot deployment can substantially reduce attacker efficiency, even if it cannot fully prevent successful detection by a well-trained attacker.

7 Discussion

This thesis investigated the interaction between Reinforcement Learning based attackers and defenders in the context of honeypot detection and deployment. Three experimental settings were explored: a single-agent attacker, a single-agent defender baseline, and a fully adversarial arena in which both agents act simultaneously on a shared environment. This section discusses the main findings, situates them in the broader research context, and reflects on their implications and limitations.

7.1 Interpretation of Results

The experimental results demonstrate that a Reinforcement Learning based attacker can learn to reliably detect honeypots under partial observability and budget constraints. In the single-agent setting, the attacker achieves high classification accuracy, indicating that the graph-based observation and probing strategy provide sufficient detail to distinguish real hosts from honeypots when no adaptive countermeasures are present.

When introducing an adaptive defender in the adversarial arena, attacker performance is substantially degraded. Average classification accuracy drops to approximately 75%, compared to over 90% in the isolated setting. This reduction confirms that an RL-based defender is capable of meaningfully interfering with the attacker's decision process by dynamically modifying host signatures and increasing ambiguity in probe responses. The defender therefore succeeds in its primary objective of reducing attacker efficiency and reliability.

However, despite this degradation, the attacker remains dominant. An average accuracy well above random chance (50%) indicates that the attacker still extracts sufficient information to correctly classify most hosts, even under active defensive interference. This asymmetry suggests that the adversarial interaction does not converge to a balanced equilibrium in which neither agent has a clear advantage. Instead, it converges to a regime where the attacker retains dominance over the final outcome (correct classification).

This distinction between outcome dominance and resistance is critical. From a defensive perspective, success does not require complete prevention of honeypot detection. Rather, increasing uncertainty, delaying classification, and forcing attackers to expend additional resources already constitute meaningful defensive gains. The arena results show that adaptive honeypot deployment can substantially reduce attacker effectiveness without fully neutralising a well-trained adversary.

7.2 Emergent Adversarial Dynamics

The adversarial arena reveals several qualitative dynamics that are not observable in the single-agent experiments. First, classification accuracy exhibits higher variance across episodes, suggesting that the defender's actions create dynamic conditions that the attacker cannot fully anticipate. Second, episodic returns indicate that misclassifications are often coupled with inefficient probing behaviour, implying that the defender not only induces errors but also disrupts the attacker's exploration–exploitation balance.

Importantly, the zero-sum reward coupling in the arena directly aligns defender incentives with attacker failure. This leads to a form of co-adaptation in which the defender learns to exploit weaknesses in the attacker's probing and classification strategy, while the attacker attempts to compensate by probing more conservatively or delaying classification. The resulting interaction resembles a cat-and-mouse dynamic rather than convergence to a fixed strategy, highlighting the suitability of multi-agent Reinforcement Learning for studying evolving cyber-security threats.

7.3 Implications for Honeypot Design

The findings of this thesis suggest that static honeypot configurations are insufficient against adaptive attackers, but that fully eliminating honeypot detection may also be unrealistic. Instead, adaptive honeypot deployment should be viewed as a means of increasing attacker uncertainty and operational cost rather than as a binary success–failure mechanism.

The defender's ability to significantly reduce attacker accuracy without direct knowledge of the attacker's policy supports the use of Reinforcement Learning for autonomous cyber deception. By continuously adjusting timing behaviour, banners, OS fingerprints, and artefact probabilities, the defender can maintain honeypots that remain plausibly realistic over time, even in the presence of learning-based attackers.

At the same time, the persistence of attacker dominance highlights the importance of combining adaptive honeypots with other defensive measures. Honeypots should therefore be considered one component in a layered defence strategy rather than a standalone solution.

7.4 Limitations

Several limitations of this work should be acknowledged. First, the environment relies on a simulated abstraction of network behaviour and honeypot artefacts. While this abstraction is sufficient to demonstrate proof-of-concept adversarial dynamics, real-world networks exhibit significantly more complexity, noise, and heterogeneity. As a result, absolute performance values should not be interpreted as direct indicators of real-world effectiveness.

Second, both agents are trained using fixed reward structures that encode specific assumptions about attacker and defender objectives. Different reward designs could lead to different strategies, particularly for the defender. Additionally, both agents use the same PPO-based training framework and similar network architectures, which may bias the interaction dynamics. Moreover, the PPO hyperparameters were selected based on commonly used values in the literature and were kept fixed across all experiments. No systematic hyperparameter tuning was performed. Consequently, the absolute performance of the agents may be suboptimal, as PPO performance is known to be sensitive to hyperparameter choices. However, since identical hyperparameter settings were used throughout, the comparative results and observed trends remain valid.

Finally, the attacker and defender are not co-trained in a fully iterative self-play regime. Each agent is evaluated against a fixed counterpart, which limits the extent to which long-term co-evolutionary dynamics can be observed.

7.5 Future Work

Future research could extend this work in several directions. A natural next step is to investigate iterative self-play, where attacker and defender policies are alternately retrained against each other to study whether more balanced equilibria emerge. Another promising direction is to introduce multiple attackers or heterogeneous attacker models, reflecting the diversity of real-world adversaries.

Further realism could be achieved by integrating richer network simulations or hybrid simulation–emulation environments, allowing the defender to operate under more realistic constraints. Finally, exploring alternative learning paradigms, such as population-based training or opponent modelling, could provide deeper insights into strategic adaptation in adversarial cyber-security settings.

Future work could also investigate the impact of systematic hyperparameter tuning on convergence speed and final performance, particularly in the adversarial setting where PPO is sensitive to parameters such as the learning rate, clipping range, and entropy regularization.

7.6 Summary

In summary, this thesis demonstrates that Reinforcement Learning provides a powerful framework for modelling both honeypot detection and adaptive honeypot deployment. The adversarial arena results show that while an RL-based attacker remains capable of detecting honeypots, an RL-based defender can substantially reduce attacker effectiveness. Rather than producing a balanced stalemate, the interaction yields a nuanced outcome in which attacker dominance is tempered by meaningful defensive resistance, offering valuable insights into the design of adaptive cyber deception systems.

8 Conclusion

This thesis investigates the application of Reinforcement Learning (RL) to honeypot detection and adaptive honeypot deployment in networked environments. By modeling both attackers and defenders as learning agents within a graph-based abstraction of an enterprise network, this work addresses the central research question: *How effectively can an RL-based attacker detect honeypots, and how can an RL-based defender adaptively deploy honeypots to minimize detection?*

Three distinct experimental settings were employed to explore the central question and its sub-questions:

8.1 Sub-Question 1

The first experiment focused on training a single-agent attacker to classify hosts in a network environment with partial observability and budget constraints. The results demonstrated that the RL-based attacker could successfully learn to probe and classify devices, achieving high classification accuracy and stable episodic returns when no adaptive countermeasures were in place. This indicates that Reinforcement Learning is a promising approach for modeling honeypot detection, as the agent was able to adapt its behavior and optimize its probing strategies.

8.2 Sub-Question 2

The second experiment examined the performance of a single-agent defender trained using Reinforcement Learning. The defender was evaluated based on conventional detection heuristics, such as those focused on timing or signature-based analysis. The results revealed that the PPO-trained defender significantly outperformed a random baseline, effectively reducing the detectability of honeypots while maintaining realistic behavior for legitimate devices. This experiment supports the idea that Reinforcement Learning allows for dynamic and adaptive honeypot deployment strategies, which outperform static configurations. The defender agent learned to adjust the deployment of honeypots to better evade detection by the attacker, showcasing the adaptability of RL-based defenders.

8.3 Sub-Question 3

The third experiment involved an adversarial setting in which both the RL-based attacker and defender interacted within the same environment. In this setup, the attacker's performance was significantly impacted compared to when it was trained in isolation. The classification accuracy of the attacker dropped to approximately 75%, and its episodic returns were reduced, indicating that the defender's adaptive deployment strategies successfully disrupted the attacker's decision-making process. While the attacker was still able to maintain classification accuracy above random chance, the interaction revealed that adaptive defense strategies increase uncertainty and the cost for attackers, forcing them to expend additional resources. These findings highlight that while the attacker retains a degree of effectiveness, the defender can meaningfully degrade the efficiency of honeypot detection.

8.4 Answering the Central Research Question

Taken together, these results demonstrate that **honeypot detection and deployment can be framed as an adversarial Reinforcement Learning problem**, where both attackers and defenders can adaptively learn and adjust their strategies. The interactions between the RL-based attacker and defender do not converge to a trivial equilibrium where one agent completely dominates the other. Instead, the system exhibits a nuanced dynamic in which attackers remain capable of detecting honeypots, but defenders can significantly reduce the accuracy and reliability of the attacker's decisions.

This suggests that adaptive honeypots should not be viewed as a binary mechanism that either succeeds or fails. Rather, their value lies in their ability to increase uncertainty, delay correct decisions, and force attackers to invest more resources into detection, ultimately raising the cost for the adversary. Thus, the use of adaptive honeypots can be a valuable component of a broader cybersecurity strategy, especially when used to introduce additional complexity and unpredictability into the attacker's decision-making process.

In conclusion, this thesis contributes a unified experimental framework for studying honeypot detection and deployment using Reinforcement Learning, demonstrates the feasibility of both attacker and defender learning agents, and provides empirical evidence that adaptive defense can meaningfully counter learning-based attackers. These results support the broader view that future cybersecurity systems must themselves be adaptive, capable of responding to evolving adversaries rather than relying on static assumptions about attacker behavior.

References

[Anu24] S. Anuyah. Understanding graph data structures: A comprehensive study. *arXiv*, 2024. Defines graphs as abstract data structures consisting of nodes (vertices) connected by edges that represent relationships between entities.

[FACU21] Javier Franco, Ahmet Aris, Berk Canberk, and A. Selcuk Uluagac. A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems. *CoRR*, abs/2108.02287, 2021.

[FDFV17] Wenjun Fan, Zhihui Du, David Fernandez, and Victor A Villagra. Enabling an anatomic view to investigate honeypot systems: A survey. 2017.

[GKCS23] Jose Antonio Gomez Gaona, Elie Kfoury, Jorge Crichigno, and Gautam Srivastava. A survey on network simulators, emulators, and testbeds used for research and education. *Computer Networks*, 2023. Discusses the use of simulation and emulation platforms to model and test network behavior for research and experimentation.

[GLC$^+$23] Chongqi Guan, Heting Liu, Guohong Cao, Sencun Zhu, and Thomas La Porta. Honeyiot: Adaptive high-interaction honeypot for iot devices through reinforcement learning. 2023.

[GZCLP25] Chongqi Guan, Jiahe Zhang, Guohong Cao, and Thomas F. La Porta. Learning-based internet of things honeypots for cyber deception. *IEEE Security and Privacy*, 2025. Describes adaptive honeypots that leverage reinforcement learning to engage and deceive attackers.

[HZ19] Linan Huang and Quanyan Zhu. Adaptive honeypot engagement through reinforcement learning of semi-markov decision processes. *CoRR*, abs/1906.12182, 2019.

[IDSM23] Niclas Ilg, Paul Duplys, Dominik Sisejkovic, and Michael Menth. A survey of contemporary open-source honeypots, frameworks, and tools. *Journal of Network and Computer Applications*, 220:103737, 2023.

[Iye21] Kumrashan Indranil Iyer. Adaptive honeypots: Dynamic deception tactics in modern cyber defense. *International Journal of Science and Research Archive*, 2021. Discusses how static honeypots become less effective against sophisticated attackers and introduces adaptive honeypot strategies as a response.

[JJT$^+$24] Amir Javadpour, Forough Ja'fari, Tarik Taleb, Mohammad Shojafar, and Chafika Benzaïd. A comprehensive survey on cyber deception techniques to improve honeypot performance. *Computers Security*, 140:103792, 2024.

[JS11] Yogendra Jain and Singh Surabhi. Honeypot based secure network system. *International Journal on Computer Science and Engineering*, 3, 02 2011.

[LWS23] Songsong Liu, Shu Wang, and Kun Sun. Enhancing honeypot fidelity with real-time user behavior emulation. 06 2023.

[LZL$^+$25] Zixuan Liu, Yi Zhao, Zhuotao Liu, Qi Li, Chuanpu Fu, Guangmeng Zhou, and Ke Xu. A hard-label black-box evasion attack against ml-based malicious traffic detection systems. *arXiv*, 2025. Demonstrates the use of reinforcement learning to adapt attack strategies to evade machine-learning-based detection systems.

[LZW25] Changsong Li, Ning Zhao, and Hao Wu. Multiple deception resources deployment strategy based on reinforcement learning for network threat mitigation. *Scientific Reports*, 15(1):16830, 2025.

[MC21] Wojciech Mazurczyk and Luca Caviglione. Cyber reconnaissance techniques. *Communications of the ACM*, 64, 02 2021.

[MDR25] Zlatan Morić, Vedran Dakić, and Damir Regvart. Advancing cybersecurity with honeypots and deception strategies. *Informatics*, 12(1), 2025.

[MSA$^+$22] Mohamed Msaad, Shreyas Srinivasa, Mikkel Andersen, David Audran, Charity Uche Orji, and Emmanouil Vasilomanolakis. Honeysweeper: Towards stealthy honeytoken fingerprinting techniques. 11 2022.

[QMMF22] Tony Quertier, Benjamin Marais, Stéphane Morucci, and Bertrand Fournel. Merlin: Malware evasion with reinforcement learning. *arXiv preprint arXiv:2203.12980*, 2022. Accessed: 2025-12-14.

[Res20] ResearchGate. Environment design for reinforcement learning: A practical guide and overview. 2020. Discusses the critical role of environment design in reinforcement learning performance and agent training.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

[SPV21] Shreyas Srinivasa, Jens Myrup Pedersen, and Emmanouil Vasilomanolakis. Gotta catch 'em all: a multistage framework for honeypot fingerprinting. *CoRR*, abs/2109.10652, 2021.

[SPV23] Shreyas Srinivasa, {Jens Myrup} Pedersen, and Emmanouil Vasilomanolakis. Gotta catch 'em all: a multistage framework for honeypot fingerprinting. *Digital Threats: Research and Practice*, 4(3), 2023.

[SWD$^+$17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[TKT$^+$24] Mark Towers, Ariel Kwiatkowski, Jordan Terry, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv*, 2024. Describes how the Gym/Gymnasium framework provides a standard API for RL environments, enabling the development, training, and evaluation of RL agents.

[URLL17] Joni Uitto, Sampsa Rauti, Samuel Laurén, and Ville Leppänen. A survey on anti-honeypot and anti-introspection methods. pages 125–134, 03 2017.

[VK22] Selvakumar Veluchamy and Ruba Soundar Kathavarayan. Deep reinforcement learning for building honeypots against runtime dos attack. *International Journal of Intelligent Systems*, 37(7):3981–4007, 2022.

[Wol25] Thomas Wolgast. Environment design for reinforcement learning: A practical guide and overview. Technical report, Carl von Ossietzky Universität Oldenburg, 2025. Surveys the main components of reinforcement learning environments, including state/observation spaces, action spaces, reward functions, and episode definitions.

[WSDE09] Gérard Wagener, Radu State, Alexandre Dulaunoy, and Thomas Engel. Self adaptive high interaction honeypots driven by game theory. In Rachid Guerraoui and Franck Petit, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 741–755, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Z$^+$20] Zonghan Zhou et al. Graph neural networks: A review of methods and applications. *AI Open*, 2020. Describes how Graph Neural Networks operate on graph structured data and capture relational patterns through message passing between nodes.

[ZLC$^+$21] Yufeng Zhang, Wanwei Liu, Zhenbang Chen, Ji Wang, and Kenli Li. On the properties of kullback–leibler divergence between multivariate gaussian distributions. *arXiv*, 2021. Analyzes KL divergence between Gaussian distributions and discusses its use as a measure of difference between distributions.

[ZT21] Li Zhang and Vrizlynn.L.L. Thing. Three decades of deception techniques in active cyber defense - retrospect and outlook. *Computers amp; Security*, 106:102288, July 2021.