School of Computer Science
Faculty of Science and Engineering
University of Nottingham
Malaysia

UG FINAL YEAR DISSERTATION REPORT

An LSTM-Based Actor-Critic Neural Network with Proximal Policy Optimisation Agent for Automated Network Penetration Testing

Student's Name	: Anay Praveen
Student Number	: 20509910
Supervisor Name	: El Ioini Nabil
Year	: 2025

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF BACHELOR OF SCIENCE IN COMPUTER SCIENCE (HONS)
THE UNIVERSITY OF NOTTINGHAM

University of Nottingham
UK | CHINA | MALAYSIA

An LSTM-Based Actor-Critic Neural Network with Proximal Policy Optimisation Agent for Automated Network Penetration Testing

Submitted in May 2025, in partial fulfillment of the conditions of the award of the degrees B.Sc.

Anay Praveen
School of Computer Science
Faculty of Science and Engineering
University of Nottingham
Malaysia

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature _____

Date _____ / _____ / _____

Acknowledgement

I would like to express my sincere thanks to everyone who helped me complete this dissertation.

First and foremost, I am deeply grateful to my supervisor, Dr El Ioini Nabil. His guidance was truly valuable throughout this journey. His expert knowledge helped me understand difficult concepts in cybersecurity. Dr Nabil was always ready to meet with me and offer constructive feedback that shaped my research direction and improved my technical approach. His constant encouragement kept me going during this challenging process and was essential to the quality of my final work.

I also wish to thank the School of Computer Science at the University of Nottingham Malaysia. The school provided an excellent learning environment that greatly supported my research efforts. The access to technical resources and relevant academic materials was crucial for carrying out my experiments. Without these facilities and support, I would not have been able to properly test and validate my research findings, which were central to this dissertation.

My sincere thanks also go to my fellow colleagues. The many discussions and brainstorming sessions we had were extremely helpful. Their different perspectives often led to new ideas, helped me solve technical problems, and improved my research methods. The feedback I received during our review sessions greatly enhanced the clarity and structure of my dissertation. These collaborations truly enriched both my research process and results.

Finally, I want to thank my family from the bottom of my heart. Their unwavering support, patience and constant encouragement have been the foundation of my academic journey. They understood the demands of this work, and their belief in me was a vital source of motivation, especially during difficult periods. This achievement reflects their support as much as my own work.

Thank you to everyone who contributed, directly or indirectly, to the completion of this dissertation.

Abstract

Contemporary cyber threats, marked by their increasing complexity and dynamism, pose significant challenges to traditional network security practices. Manual penetration testing methods, often resource-intensive and performed periodically, struggle to deliver the continuous, comprehensive security validation demanded by modern, intricate network infrastructures. This limitation highlights a pressing need for more efficient, adaptive, and automated approaches to proactive security assessment.

This dissertation addresses this need by developing and evaluating a novel reinforcement learning agent for automated network penetration testing. The agent employs a Long Short-Term Memory (LSTM) based Advantage Actor-Critic (A2C) architecture, leveraging LSTM's capacity to process sequential observations and manage partial observability inherent in complex network environments. Policy learning is stabilised using the Proximal Policy Optimisation (PPO) algorithm. Crucially, a structured curriculum learning framework is implemented within the Network Attack Simulator (NASim) environment, enabling the agent to progressively acquire skills by training on scenarios of increasing difficulty.

The research investigates the feasibility and effectiveness of this integrated LSTM-PPO and curriculum learning strategy. Experimental results demonstrate that the developed agent significantly outperforms established NASim benchmarks across diverse network topologies. It achieves high success rates, often reaching 100%, while requiring substantially fewer steps and obtaining higher rewards, indicating superior efficiency and strategy development. The findings also underscore the critical influence of network topology on task difficulty and validate the effectiveness of curriculum learning for knowledge transfer and mastering complex scenarios. This work establishes the proposed approach as a viable and potent method for creating autonomous agents capable of intelligently navigating networks and identifying vulnerabilities with minimal human intervention, offering a significant advancement in automated security assessment.

Table of Contents

List of Figures v

List of Tables vi

1 Introduction 1
- 1.1 Motivation . . . 1
  - 1.1.1 Escalating Complexity of Cyber Threats . . . 2
  - 1.1.2 Limitations of Traditional Penetration Testing . . . 2
  - 1.1.3 Advances in Reinforcement Learning for Cybersecurity . . . 3
  - 1.1.4 Need for Efficient and Adaptive Security Assessment . . . 3
- 1.2 Aim . . . 4
- 1.3 Objectives . . . 4

2 Related Work 5
- 2.1 Traditional Penetration Testing . . . 5
- 2.2 Reinforcement Learning for Cybersecurity . . . 5
- 2.3 Advanced Reinforcement Learning Architectures . . . 7
- 2.4 Curriculum Learning and Transfer in Security Assessment . . . 8
- 2.5 Simulation Environments and Attack Modelling . . . 9
- 2.6 Advanced Attack Strategies and Specialised Applications . . . 10
- 2.7 Ethical Considerations and Responsible Research . . . 10

3 Methodology 12
- 3.1 System Architecture . . . 12
- 3.2 LSTM-Based Sequential Processing . . . 13
  - 3.2.1 LSTM Cell Formulation . . . 14
  - 3.2.2 Observation Processing Pipeline . . . 14
  - 3.2.3 Sequence Length Optimisation . . . 15
- 3.3 Advantage Actor-Critic Framework . . . 15
  - 3.3.1 Value Network (Critic) . . . 16
  - 3.3.2 Policy Network (Actor) . . . 17
  - 3.3.3 Advantage Estimation . . . 17
  - 3.3.4 Specialised Advantage Function for Penetration Testing . . . 17
- 3.4 Proximal Policy Optimisation Update . . . 18
  - 3.4.1 Clipped Surrogate Objective Function . . . 19
  - 3.4.2 Combined Loss Function . . . 19
- 3.5 Curriculum Learning Framework . . . 20
  - 3.5.1 Network Scenario Progression . . . 20
  - 3.5.2 Knowledge Transfer Mechanisms . . . 21
- 3.6 Reward Function Design . . . 21
- 3.7 Training Process . . . 22
  - 3.7.1 Hyperparameter Selection . . . 22
  - 3.7.2 Adaptive Hyperparameter Scheduling . . . 22
- 3.8 Experimental Setup . . . 23
  - 3.8.1 NASim Environment . . . 23
  - 3.8.2 Partial Observability . . . 24
  - 3.8.3 Action Space . . . 24
  - 3.8.4 Training Schedule . . . 25

3.8.5 Evaluation Metrics 26

4 Results **27**
4.1 Performance During Evaluation 27
4.2 Final Performance Across Scenarios and Comparison to Benchmark . . 29
4.3 Analysis of Network Topology Impact 31
4.4 Curriculum Learning Effectiveness 32

5 Discussion **33**
5.1 Analysis of Performance Patterns 33
5.2 Implications for Automated Penetration Testing 34
5.3 Generalisability Considerations 34
5.4 Limitations and Challenges 35
5.5 Future Directions 35

6 Conclusion **37**

7 References **38**

8 Appendix **41**
8.1 Software and Tools Used 41
8.2 System Specifications 41
8.3 Implementation Details 42
8.3.1 Observation Processing & LSTM Input 42
8.3.2 Network Architectures (LSTM & A2C) 42
8.3.3 Action Selection & Masking 42
8.3.4 Advantage Estimation (GAE + Progress Term) 42
8.3.5 PPO-Style Update Mechanism 43
8.3.6 Knowledge Transfer 43
8.4 Training Results Visualizations 43

List of Figures

1 LSTM-based Actor-Critic Agent Architecture for Network Penetration Testing (using PPO-style updates) 12
2 Long Short-Term Memory (LSTM) Network Cell Structure 13
3 LSTM Integrated with Advantage Actor-Critic (A2C) 16
4 Advantage Actor-Critic (A2C) Framework 16
5 Advantage Actor-Critic (A2C) Framework with PPO-style Update Mechanism 18
6 Proximal Policy Optimisation (PPO) Core Concept 18
7 Reward performance curves for all scenarios, showing mean reward (± std dev) measured during evaluation episodes throughout testing. . . . 27
8 Steps-per-episode performance curves for all scenarios, showing mean steps (± std dev) measured during evaluation episodes throughout testing. 28
9 Final Success Rate of the LSTM-based Actor-Critic Agent across all network scenarios after training. 29
10 Comparison of Developed Agent vs. Benchmark Agent Performance. Left: Mean Steps per Episode (Lower is better). Right: Mean Rewards per Episode (Higher is better). 30
11 Performance metrics for the 'tiny' scenario. 43
12 Performance metrics for the 'tiny-hard' scenario. 43
13 Performance metrics for the 'tiny-small' scenario. 44
14 Performance metrics for the 'small' scenario. 44
15 Performance metrics for the 'small-honeypot' scenario. 44
16 Performance metrics for the 'small-linear' scenario. 44
17 Performance metrics for the 'medium' scenario. 44
18 Performance metrics for the 'medium-single-site' scenario. 45
19 Performance metrics for the 'medium-multi-site' scenario. 45

List of Tables

1 Performance Comparison Across LSTM Sequence Lengths 15
2 Curriculum Stages for Network Penetration Testing 20
3 Core Hyperparameters for Agent Training 23
4 Scenario-Specific Hyperparameter Settings (Overrides) 23
5 NASim Benchmark Scenario Configurations 24
6 Action Space Categories for Network Penetration Testing 25
7 Training Schedule per Scenario in Curriculum 25
8 NASim Benchmark Scenario Agent Performance 26
9 Trained Agent Final Performance Metrics Across Network Scenarios (Mean $\pm$ Stdev over 100 eval episodes) 29

1 Introduction

Today's digital landscape features a vast proliferation of interconnected systems, cloud infrastructure, and networks underpinning global commerce and society. This rapid expansion in connectivity significantly enlarges the attack surface for malicious actors, presenting complex cybersecurity challenges beyond traditional approaches. Consequently, organisations face the difficult task of securing extensive network infrastructures against attackers who require just a single vulnerability to succeed.

Penetration testing has emerged as a cornerstone of proactive cybersecurity strategies, enabling organisations to identify security weaknesses through controlled exploitation of vulnerabilities [1]. However, traditional methodologies are constrained by their reliance on manual processes, domain-specific expertise, and point-in-time assessments that fail to address the dynamic nature of contemporary threat landscapes.

The integration of artificial intelligence and machine learning technologies into penetration testing workflows represents a promising approach to overcoming these limitations [2, 3]. Among various machine learning paradigms, reinforcement learning has demonstrated particular efficacy in addressing the sequential decision-making processes that characterise penetration testing scenarios [4, 5]. By formulating penetration testing as a reinforcement learning problem, autonomous agents have been developed capable of navigating complex network environments, identifying vulnerability chains, and executing exploitation strategies with minimal human intervention [6–8].

The Network Attack Simulator (NASim) provides a configurable, scalable environment for simulating network penetration scenarios [6, 9], enabling the training and evaluation of autonomous agents across diverse network topologies and vulnerability distributions. Despite progress in applying reinforcement learning to penetration testing, significant challenges remain in developing agents capable of navigating the complexity, partial observability, and high-dimensional state spaces that characterise real-world networks [3, 10].

This dissertation introduces a Long Short-Term Memory (LSTM) and Proximal Policy Optimization (PPO) agent specifically designed for network penetration testing across diverse network topologies. The proposed approach leverages the sequential decision-making capabilities of LSTM networks [11] and the policy optimisation strengths of PPO [12] within an Advantage Actor-Critic (A2C) framework [13] to address the partial observability, high-dimensional state spaces, and long-term dependencies that characterise real-world penetration testing scenarios [7, 14].

1.1 Motivation

The development of an development of an agent employing a Long Short-Term Memory (LSTM) based Actor-Critic architecture with Proximal Policy Optimisation (PPO) updates for network penetration testing is motivated by several interconnected factors that collectively underscore the urgent need for more efficient, adaptive, and comprehensive security assessment methodologies.

1.1.1 Escalating Complexity of Cyber Threats

The evolution of cyber threats from simplistic exploitation techniques to sophisticated, multi-stage attack campaigns has fundamentally transformed the cybersecurity landscape. Modern adversaries employ complex attack methodologies that leverage multiple vulnerabilities across distributed systems, utilise sophisticated evasion techniques, and adapt dynamically to defensive countermeasures.

The SolarWinds supply chain attack of 2020 exemplifies this complexity, where attackers compromised the software build system to inject malicious code into legitimate software updates, ultimately affecting approximately 18,000 organisations globally. Similarly, the Colonial Pipeline ransomware attack in 2021 highlighted the cascading impacts of cyber attacks on critical infrastructure, causing fuel shortages across the eastern United States.

The asymmetric nature of cybersecurity compounds this challenge, as defenders must secure all potential entry points whilst attackers need only identify a single exploitable vulnerability. This asymmetry is particularly evident in network security contexts, where the expansion of network perimeters through cloud integration, remote work infrastructure, and Internet of Things (IoT) deployments has created sprawling attack surfaces.

The polymorphic nature of modern malware further amplifies this challenge, with advanced threats capable of modifying their signatures, behaviours, and communication patterns to evade detection. Furthermore, the time-sensitive nature of vulnerability exploitation creates additional pressure on security teams, with the average time between vulnerability discovery and exploitation decreasing significantly in recent years.

The weaponisation of artificial intelligence for offensive purposes introduces yet another dimension to the evolving threat landscape. Malicious actors are increasingly employing machine learning techniques to enhance attack capabilities, necessitating corresponding advancements in defensive technologies and methodologies [7, 8, 15].

1.1.2 Limitations of Traditional Penetration Testing

Traditional penetration testing methodologies, whilst valuable for identifying specific security vulnerabilities, suffer from several inherent limitations that diminish their effectiveness in addressing contemporary cybersecurity challenges.

From an operational perspective, manual penetration testing is inherently labour-intensive and time-consuming, requiring skilled security professionals to systematically explore network environments. This manual approach creates significant resource constraints, with comprehensive penetration tests of enterprise environments typically requiring numerous person-hours and spanning multiple weeks.

The scarcity of qualified penetration testing professionals further exacerbates this operational challenge. The global cybersecurity workforce gap represents a fundamental imbalance between security demands and available expertise. The reliance on human expertise introduces inconsistent assessment outcomes, with research showing that manual penetration tests conducted by different security professionals on identical network environments often identify significantly different vulnerability sets.

Traditional penetration testing approaches are also constrained by their discrete, point-in-time nature, which fails to address the continuous evolution of both network environments and threat landscapes. The typical annual or quarterly cadence of penetration tests creates significant gaps between assessments, during which new vulnerabilities may be introduced through system changes or software updates.

Technical limitations further constrain the effectiveness of traditional approaches. The complexity and scale of modern network environments exceed the cognitive capacity of individual security professionals, leading to selective testing approaches that focus on high-priority systems whilst potentially overlooking less obvious attack vectors.

The significant cost and time involved in manual penetration testing often make comprehensive security assessments financially unviable, particularly for small and medium-sized enterprises (SMEs).

1.1.3 Advances in Reinforcement Learning for Cybersecurity

Recent advancements in reinforcement learning (RL) are significantly enhancing penetration testing by offering automated and adaptive approaches to security assessments [2, 3]. RL, focused on sequential decision-making under uncertainty, naturally models the strategic exploration and exploitation inherent in penetration testing [4, 5].

Seminal research demonstrated RL's efficacy in automated vulnerability discovery, showing improvements over traditional scanning tools [16, 17]. Deep Q Networks (DQN) have shown promise in identifying vulnerabilities in simulated environments [17, 18], with comparative studies indicating DQN agents discovered more vulnerabilities than traditional scanners in complex networks.

More advanced policy gradient methods like Advantage Actor-Critic (A2C) [13, 19], Trust Region Policy Optimization (TRPO) [20], and Proximal Policy Optimization (PPO) [12] have demonstrated superior performance [21]. These algorithms improve stability and sample efficiency, crucial in complex, partially observable network environments.

Integrating recurrent neural networks, particularly Long Short-Term Memory (LSTM) networks [11], with RL further enhances capabilities by enabling agents to leverage historical interaction data and capture long-term dependencies [7, 14, 22, 23].

Meta-learning approaches address transferability challenges by allowing agents to rapidly adapt to new environments based on prior experiences [24]. The development of specialised simulation environments like the Network Attack Simulator (NASim) [6,9] has accelerated progress by providing flexible frameworks for training autonomous agents.

1.1.4 Need for Efficient and Adaptive Security Assessment

The dynamic nature of contemporary cybersecurity threats and the continuous evolution of network infrastructures necessitate security assessment approaches that are both efficient and adaptive. Traditional methodologies, characterised by periodic evaluations, are increasingly inadequate for addressing continuously evolving security challenges.

The acceleration of software development cycles through agile methodologies and DevOps practices, with continuous deployment, leads to hundreds or thousands of

changes daily. Each deployment can introduce vulnerabilities, creating a significant gap between traditional periodic testing and the rate new attack surfaces emerge.

The shift towards cloud computing and infrastructure as code further accelerates the rate of change in modern IT environments. Cloud platforms enable rapid provisioning, scaling, and reconfiguration of resources, creating dynamic environments that evolve continuously in response to business requirements.

The increasing complexity and scale of modern network infrastructures further underscore the need for efficient security assessment methodologies. Enterprise networks encompass thousands or millions of endpoints across diverse environments (on-premises, cloud, edge, remote). This distributed architecture creates sprawling attack surfaces impractical for comprehensive manual assessment.

1.2 Aim

This dissertation aims to develop and evaluate an optimised reinforcement learning agent, employing a Long Short-Term Memory (LSTM) based Actor-Critic architecture with Proximal Policy Optimisation (PPO) updates, for conducting automated network penetration testing within diverse simulated network environments (NASim). By integrating a structured curriculum learning framework and knowledge transfer mechanisms, this study investigates the feasibility and effectiveness of this approach compared to established benchmarks. The goal is to address fundamental limitations in traditional penetration testing, enabling the creation of autonomous agents capable of comprehensive security assessments that minimise human intervention while maximising the efficacy of vulnerability detection across varied network topologies.

1.3 Objectives

The primary objectives of this dissertation are to:

1)Design and implement an optimised LSTM-A2C-PPO agent architecture for automated penetration testing.

2)Utilize LSTM for sequential decision-making, state maintenance, and identifying multi-step attack chains.

3)Implement a structured curriculum learning framework in NASim for progressive skill acquisition.

4)Develop weight transfer mechanisms for effective knowledge application across curriculum stages.

5)Evaluate agent performance across NASim scenarios and rigorously compare against established benchmarks.

2 Related Work

This section presents a comprehensive review of prior work relevant to automated network penetration testing using reinforcement learning. The examination encompasses traditional penetration testing methodologies, advancements in reinforcement learning for cybersecurity applications, specialised architectures for sequential decision-making in complex environments, curriculum learning approaches, and recent developments in simulation environments.

2.1 Traditional Penetration Testing

Traditional penetration testing has historically been a manual, expert-driven process characterised by systematic exploration of target systems to identify and exploit vulnerabilities [1]. While manual penetration testing can be thorough and contextually aware, it suffers from significant limitations in scalability, consistency, and reproducibility [2, 3, 10]. The effectiveness of traditional penetration testing is heavily dependent on the expertise and experience of individual testers, leading to substantial variability in assessment outcomes.

The labour-intensive nature of manual penetration testing creates substantial resource constraints that limit the frequency and comprehensiveness of security assessments [2, 3]. This manual approach creates temporal gaps in security coverage that can be exploited by adversaries.

Traditional penetration testing methodologies also face significant challenges in addressing the increasing complexity and scale of modern network infrastructures. This expanded attack surface renders manual exploration increasingly impractical.

The discrete, point-in-time nature of traditional penetration testing creates additional limitations in dynamic environments. This temporal discontinuity is particularly problematic in modern development environments, where DevOps practices and continuous integration/continuous deployment (CI/CD) pipelines can introduce frequent system changes.

The economics of manual penetration testing represent another significant constraint, with high costs limiting access to comprehensive security assessments. This economic constraint is particularly problematic for small and medium-sized organisations with constrained security budgets.

These multifaceted limitations of traditional penetration testing methodologies underscore the need for more automated, scalable, and consistent approaches to security assessment that can address the complex and dynamic nature of contemporary cybersecurity challenges.

2.2 Reinforcement Learning for Cybersecurity

Reinforcement learning (RL) has emerged as a promising paradigm for automating cybersecurity tasks, offering the potential to address many limitations of traditional approaches [2, 3]. The fundamental alignment between reinforcement learning principles and penetration testing processes creates natural opportunities for integration. As explained by Ghanem and Chen [3], reinforcement learning provides a mathematical framework for sequential decision-making under uncertainty,

which closely mirrors the incremental exploration and exploitation processes that characterise penetration testing [4, 5].

Seminal work by Elderman et al. established early foundations for applying reinforcement learning to network security, demonstrating that relatively simple Q-learning agents could discover attack paths in simulated network environments [16]. Their experiments showed promising results in small network topologies but highlighted limitations in scalability to more complex environments. Building on this foundation, Hu et al. expanded the application of reinforcement learning to more realistic network security scenarios, implementing Deep Q-Networks (DQN) that significantly outperformed tabular methods in terms of both learning efficiency and policy quality [17]. Other advancements in value-based methods include Dueling Network architectures [25].

The integration of deep learning with reinforcement learning has substantially enhanced the capabilities of automated security assessment systems. Li et al. [26] demonstrated the efficacy of deep neural network function approximators for security auditing tasks, achieving significant improvements in vulnerability identification compared to traditional scanning tools. Their work highlighted the potential of deep reinforcement learning to enhance the efficiency and effectiveness of security assessments through intelligent exploration strategies and adaptive exploitation techniques.

Recent research by Hausknecht and Stone [14] has focused on addressing the partial observability challenges inherent in penetration testing scenarios. They introduced a framework leveraging recurrent neural networks with Deep Recurrent Q-Networks (DRQN) to maintain state information across time steps, enabling more effective decision-making in environments where the agent has limited visibility of the network state. Their experimental results demonstrated that DRQN agents achieved significantly higher success rates than standard DQN agents in penetration testing scenarios with partial observability, highlighting the importance of architectural choices that address the specific characteristics of security assessment environments.

Schwartz and Kurniawati made significant contributions through their Network Attack Simulator (NASim) [6, 9], which provides a configurable, scalable environment for simulating network penetration scenarios and training autonomous agents. NASim supports partial observability, action spaces that reflect real-world penetration testing tools, and reward structures that align with security assessment objectives. This framework has become an important platform for developing and evaluating reinforcement learning approaches to network penetration testing, enabling researchers to train and evaluate agents across diverse network topologies and vulnerability distributions. The introduction of enhanced simulation environments that combine simulation and emulation capabilities addresses the reality gap in simulation-based training, enabling the validation of reinforcement learning agents in more realistic environments [27].

The application of reinforcement learning to penetration testing extends beyond vulnerability discovery to include strategy development and attack planning. Liu et al. proposed a hierarchical reinforcement learning approach for automated penetration testing, decomposing the complex penetration testing process into multiple levels of subtasks [28]. Their hierarchical architecture enabled more efficient learning and better generalisation across different network environments, with agents learning to select high-level strategies before determining specific exploitation techniques.

Autonomous network defence using reinforcement learning is also an active area of research, exploring the use of RL agents for defensive strategies [29]. Furthermore, offline reinforcement learning techniques [30], which learn from static datasets, present potential for leveraging existing penetration testing data to train agents.

2.3 Advanced Reinforcement Learning Architectures

Recent advancements in reinforcement learning architectures have significantly enhanced the capabilities of automated penetration testing systems. Particularly notable is the development of policy gradient methods such as Proximal Policy Optimisation (PPO), which offer advantages in terms of stability and sample efficiency compared to earlier approaches [12].

Udupa et al. conducted a comparative analysis of reinforcement learning algorithms for network penetration testing, evaluating the performance of DQN, Advantage Actor-Critic (A2C) [13, 19], Trust Region Policy Optimisation (TRPO) [20], and PPO across diverse network scenarios [15]. Their results demonstrated that PPO consistently outperformed other algorithms, achieving higher success rates and more efficient exploitation paths. This superior performance was attributed to PPO's clipped objective function, which constrains policy updates to prevent catastrophic performance degradation while still allowing significant improvements. Their work established PPO as a particularly effective algorithm for penetration testing applications, capable of navigating the complex, partially observable environments that characterise real-world networks. Deterministic Policy Gradient (DPG) algorithms [31, 32] are also a key class of policy gradient methods, often applied to continuous control problems but conceptually related to the policy optimisation task.

Memory-augmented reinforcement learning architectures have shown particular promise for penetration testing scenarios, where the optimal action at any given time often depends on the sequence of previous actions and observations. Yi and Liu implemented various recurrent neural network architectures for penetration testing agents, comparing the performance of Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Transformer-based models [7]. Their experimental results demonstrated that LSTM-based agents achieved high success rates in complex penetration testing scenarios, attributed to LSTM's ability to capture long-term dependencies in sequential decision processes [11, 23] while effectively managing the vanishing gradient problem. Further work has explored recurrent networks like DRQN for partial observability [14, 22].

The integration of attention mechanisms with reinforcement learning architectures has further enhanced the capability of automated penetration testing systems to focus on relevant aspects of complex network states. Yang and Liu developed an attention-augmented architecture for network penetration testing, enabling agents to selectively attend to critical network features when making decisions [33]. Their approach demonstrated improvements in both learning efficiency and final performance compared to standard implementations, particularly in scenarios involving large, complex network topologies.

Multi-agent reinforcement learning approaches offer additional capabilities for simulating coordinated attacks and evaluating defensive strategies against multiple attackers. Louati et al. and Ren et al. explored cooperative multi-agent reinforcement learning for penetration testing, demonstrating that collaborative agents could discover

more sophisticated attack paths than single-agent approaches [34, 35]. Their work highlighted the potential of multi-agent systems to identify complex vulnerability chains that might be missed by individual agents.

The combination of adversarial training with reinforcement learning has shown promise for developing more robust penetration testing agents. Building on the principles of Robust Adversarial Reinforcement Learning (RARL), Sychugov and Grekov implemented an adversarial training framework for penetration testing agents, where defender agents continuously adapted their strategies to counter the attacker's exploits [36]. This co-evolutionary approach resulted in more robust penetration testing agents capable of discovering vulnerabilities even in environments with adaptive defensive measures, more closely approximating the dynamic nature of real-world security competitions. Techniques like Layer Normalisation [37] are often employed in these complex network architectures to improve training stability.

2.4 Curriculum Learning and Transfer in Security Assessment

Curriculum learning approaches have emerged as a particularly effective strategy for training reinforcement learning agents for complex penetration testing tasks. Following the foundational work of Bengio et al. [38], curriculum learning in the context of penetration testing involves progressively increasing the complexity of training scenarios, enabling agents to develop foundational knowledge before attempting sophisticated attack scenarios [39].

Wilson et al. applied curriculum learning principles to penetration testing agent development, creating a structured progression of increasingly complex network environments [39]. Their research demonstrated that agents trained through a curriculum achieved significantly higher success rates on challenging test scenarios compared to agents trained directly on difficult environments. This substantial performance difference underscores the value of progressive training approaches for developing sophisticated penetration testing agents.

Knowledge transfer mechanisms represent another important advancement for enhancing the efficiency and effectiveness of automated penetration testing. Udupa et al. explored transfer learning approaches for penetration testing, developing techniques to transfer knowledge between agents operating in different network environments [15]. Their transfer learning framework enabled agents to leverage knowledge gained from simpler scenarios when addressing more complex environments, significantly reducing the training time required to achieve high performance in new scenarios.

Meta-learning techniques, which enable agents to rapidly adapt to new environments based on previous experiences, have shown particular promise for addressing the transferability challenges in penetration testing [24]. Finn et al. implemented a model-agnostic meta-learning (MAML) approach that can be applied to network penetration testing, training agents to quickly adapt to novel network configurations [24]. Their experiments demonstrated that meta-learned agents achieved substantial performance levels after observing just a few episodes in new environments. This rapid adaptation capability addresses one of the key limitations of traditional penetration testing approaches, which typically require significant reconfiguration and manual adjustment when applied to new network environments.

The development of continual learning techniques further enhances the capability of

penetration testing agents to adapt to evolving environments and threat landscapes. Liu et al. proposed a continual learning framework for penetration testing agents, enabling adaptation to changing network configurations without catastrophic forgetting of previously learned knowledge [28]. Their approach incorporated experience replay mechanisms to maintain performance on previously encountered scenarios while adapting to new challenges. Techniques like Hindsight Experience Replay (HER) [40] have also been developed to improve sample efficiency in sparse-reward environments, which is highly relevant to penetration testing.

2.5 Simulation Environments and Attack Modelling

The development of specialised simulation environments has been crucial for advancing research in automated penetration testing. Beyond the seminal work on NASim by Schwartz and Kurniawati [6, 9], several recent developments have significantly enhanced the realism and capabilities of penetration testing simulation environments.

The integration of emulation capabilities with simulation environments represents an important advancement for bridging the reality gap in automated penetration testing [27]. Janisch et al. developed a framework that combines abstract simulation for efficient training with selective emulation of critical actions to validate their real-world feasibility. This hybrid approach enables more efficient agent training while ensuring that learned policies remain applicable to real-world environments.

Recent work has also focused on incorporating more realistic vulnerability models into penetration testing simulations. These frameworks model the stochastic nature of real-world exploits, including success probabilities, execution times, and potential for system crashes or instability, enabling more realistic evaluation of penetration testing strategies.

The modelling of adversarial behaviours and defensive responses represents another important dimension of penetration testing simulation. Building on the principles of Robust Adversarial Reinforcement Learning (RARL), Sychugov and Grekov implemented an adaptive defender model for penetration testing simulations [36]. Their framework enables the simulation of realistic defensive responses to detected intrusion attempts, including alert generation, connection termination, and dynamic reconfiguration of security controls. This adaptive defender model creates more challenging and realistic training environments for penetration testing agents.

The development of scenario generation techniques represents another significant advancement for training robust penetration testing agents. Nguyen et al. proposed a procedural generation framework for penetration testing scenarios with PenGym, enabling the automatic creation of diverse network configurations with varying topologies, vulnerability distributions, and security controls [41]. This approach addresses the limited diversity of manually created training scenarios, enabling more comprehensive evaluation of agent performance across the spectrum of potential real-world environments.

2.6 Advanced Attack Strategies and Specialised Applications

Recent research has extended reinforcement learning applications in penetration testing to specialised security domains and advanced attack methodologies. These developments highlight the versatility of reinforcement learning approaches and their potential to address diverse security challenges. Research by Yang and Liu exploring curiosity-driven and multi-objective approaches [33] aims to generate more diverse and sophisticated attack behaviours.

The application of reinforcement learning to privilege escalation vulnerabilities represents an important development in post-exploitation capabilities. Recent work on the Raijū framework has implemented specialized reinforcement learning agents using both A2C and PPO to automatically launch privilege escalation attacks using Metasploit modules [41]. Their approach demonstrates how RL can identify and exploit vulnerabilities to gain elevated privileges within compromised systems.

The application of reinforcement learning to social engineering and phishing attacks represents another emerging research direction. Recent studies have proposed AI-based frameworks for assessing vulnerability to sophisticated phishing campaigns, enabling security teams to evaluate and mitigate targeted social engineering attacks [42]. These approaches incorporate behavioral analytics to identify potential vulnerabilities before attacks occur.

Research on web application security has also benefited from reinforcement learning techniques. Li et al. developed the INNES model, a deep reinforcement learning-based approach for network penetration testing that can identify and exploit vulnerabilities in networked applications [26]. Their work demonstrated how reinforcement learning agents can discover complex injection vulnerabilities and authentication weaknesses.

The development of evasion techniques to bypass intrusion detection systems represents another important application of reinforcement learning in penetration testing. Javaid et al. and Chung et al. explored deep learning approaches for network intrusion detection systems, which provide the foundation for developing adversarial evasion techniques [42, 43]. These frameworks enable the development of sophisticated obfuscation strategies that can evade detection systems.

Recent work has also explored hardware-level security verification using reinforcement learning. Hardware security frameworks apply reinforcement learning to system-on-chip (SoC) security verification, enabling the identification of hardware vulnerabilities without requiring detailed internal knowledge of the system. The use of universal approximation theorems for neural networks [44] provides a theoretical basis for the ability of deep RL agents to learn complex attack strategies.

2.7 Ethical Considerations and Responsible Research

The development of automated penetration testing systems raises important ethical considerations regarding potential misuse and responsible research practices. The dual-use nature of these technologies necessitates careful attention to ethical frameworks and responsible disclosure protocols.

Ghanem and Chen conducted a comprehensive analysis of the ethical implications of automated penetration testing research, examining potential risks and mitigation strategies [3, 10]. Their work highlighted the importance of establishing clear ethical

boundaries for research in this domain, particularly as AI systems become more capable of conducting sophisticated attacks autonomously.

The potential for reinforcement learning systems to discover novel attack methodologies creates particular ethical challenges. Uitto et al. examined these challenges in the context of automated vulnerability discovery [45], proposing a staged disclosure model that balances security research benefits with risk mitigation. Their approach addresses the critical need for responsible disclosure frameworks when AI systems identify previously unknown vulnerabilities.

The integration of automated penetration testing into formal security assessment frameworks represents an important step toward responsible deployment of these technologies. As highlighted in recent research, reinforcement learning-based penetration testing requires appropriate governance controls, operational safeguards, and compliance considerations to ensure that these powerful tools are used ethically [46].

Human-in-the-loop approaches represent another important direction for responsible deployment of automated penetration testing systems. Nguyen and Reddi propose a collaborative framework where reinforcement learning agents work alongside human security professionals, combining the efficiency and consistency of automated systems with the contextual understanding and judgment of human experts [41]. This approach ensures that automated systems enhance rather than replace human expertise, maintaining critical ethical oversight in the penetration testing process.

3 Methodology

This section presents a comprehensive methodology for developing an LSTM-based Actor-Critic reinforcement learning framework, utilising PPO-style updates, for automated network penetration testing. The approach addresses inherent challenges such as partial observability, sequential decision-making, and the need for efficient exploration of complex network environments. The section begins with an outline of the overall system architecture, followed by detailed descriptions of the LSTM-based sequential processing, Advantage Actor-Critic framework, and the PPO clipped objective components. Subsequently, the curriculum learning framework enabling progressive knowledge acquisition across scenarios of increasing complexity is discussed. Finally, the experimental setup used to evaluate the approach is presented.

3.1 System Architecture

The proposed system architecture integrates three key components to address the challenges of automated penetration testing: a sequential observation processor using LSTM networks, an Advantage Actor-Critic (A2C) framework for policy and value estimation, and the Proximal Policy Optimisation (PPO) clipped objective algorithm for stable policy updates [12]. Figure 1 provides a visual representation of this integrated architecture, illustrating the information flow and computational processes involved in the agent's operation.

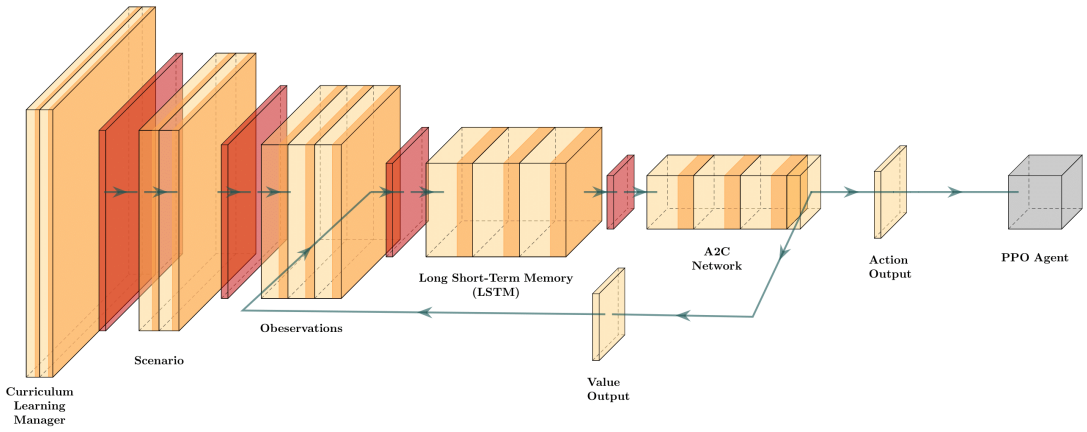


Figure 1: LSTM-based Actor-Critic Agent Architecture for Network Penetration Testing (using PPO-style updates)

As depicted in Figure 1, the system operates through a closed-loop process comprising the following steps:

1. The agent receives observations from the network environment.
2. The LSTM network processes sequences of these observations, maintaining an internal state representation.
3. The A2C networks compute action probabilities (policy) and state value estimates (value).
4. The agent selects and executes an action in the environment based on the policy.
5. The environment returns a reward signal and a new observation.

6. This information (transition) is stored and used periodically to update the agent's policy and value networks using the PPO clipped objective and GAE.

This architectural design addresses several challenges specific to penetration testing:

- **Partial Observability:** The LSTM component maintains an internal representation of the environment state across time steps by processing sequences of observations, enabling the agent to make informed decisions based on historical context [11, 14].
- **Sequential Decision-Making:** The A2C framework enables the agent to learn both state values (critic) and action policies (actor), facilitating effective exploitation of multi-step vulnerability chains [13].
- **Stable Policy Updates:** The PPO clipped objective function constrains policy updates, preventing large, destabilising changes and promoting more stable learning compared to vanilla policy gradients [12].
- **Exploration-Exploitation Balance:** Adaptive entropy regularisation, adjusted based on performance, encourages sufficient exploration of new attack paths while allowing exploitation of known vulnerabilities.

3.2 LSTM-Based Sequential Processing

Network penetration testing inherently involves sequential decision-making where the optimal action at any time step depends on both the current observation and the history of previous observations and actions. To address this temporal dependency, a Long Short-Term Memory (LSTM) network is implemented to process sequences of observations. Figure 2 illustrates the architecture of the LSTM network, demonstrating its internal structure and information flow pathways.

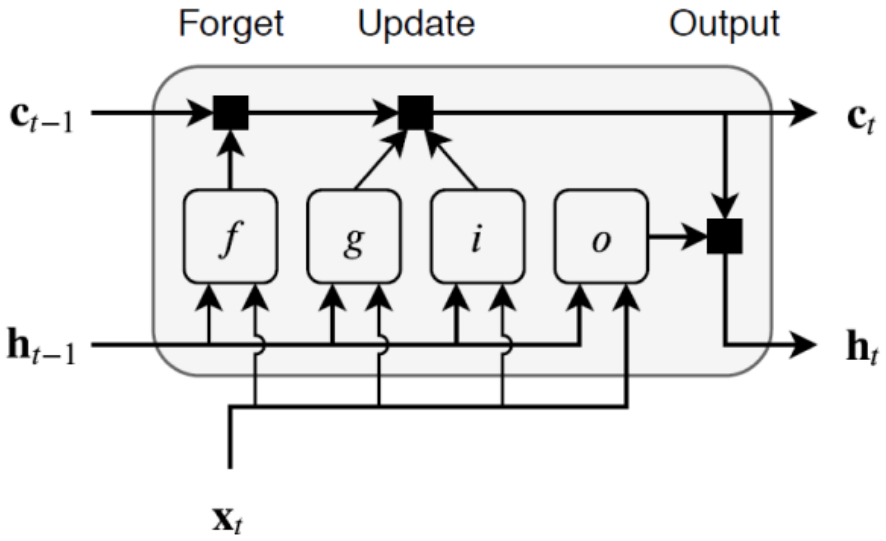


Figure 2: Long Short-Term Memory (LSTM) Network Cell Structure

3.2.1 LSTM Cell Formulation

The mathematical formulation of the LSTM cell defines its operation through a series of gating mechanisms and state updates, as expressed in the following equations:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \tag{1}$$
$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \tag{2}$$
$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \tag{3}$$
$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \tag{4}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{5}$$
$$h_t = o_t \odot \tanh(c_t) \tag{6}$$

Where these components serve the following functions:

- i_t, f_t, o_t represent the input, forget, and output gates at time step t.
- g_t is the cell input activation (candidate values) at time step t.
- c_t is the cell state at time step t.
- h_t is the hidden state (output) at time step t.
- x_t is the input at time step t.
- W and b denote the weight matrices and bias vectors for the respective gates and inputs.
- σ is the sigmoid activation function, mapping values to $[0, 1]$.
- $\tanh$ is the hyperbolic tangent activation function, mapping values to $[-1, 1]$.
- $\odot$ represents element-wise multiplication (Hadamard product).

The theoretical foundation for using activation functions such as sigmoid and tanh in neural networks is provided by universal approximation theorems [44].

The implementation maintains a sequence of recent observations, enabling the network to learn temporal patterns in the penetration testing process. This capability is particularly important for identifying multi-step attack paths that require sequential exploitation of vulnerabilities across different hosts. Additional techniques such as Layer Normalisation [37], applied within the input processing layer and potentially within the LSTM cell itself, can improve training stability.

3.2.2 Observation Processing Pipeline

The observation processing pipeline transforms raw environmental observations into a format suitable for sequential processing through the LSTM via three principal stages:

1. **Preprocessing**: Raw observations are flattened and normalised. Normalisation helps stabilise training and ensures features are on a comparable scale. A simple min-max scaling or standardisation can be used. For instance, standardisation uses:

$$x_{\text{norm}} = \frac{x - \mu}{\sigma + \epsilon} \tag{7}$$

where μ and σ are the running mean and standard deviation of the

observations, and ϵ is a small constant (e.g., 10^{-8}) to prevent division by zero. The implementation uses a clipping and scaling approach that transforms observations to a standardised range.

2. **Sequential Buffer**: Processed observations (x_{norm}) are added to a sequence buffer of fixed length L. The buffer at time t holds the sequence:

$$S_t = [x'_{t-(L-1)}, x'_{t-(L-2)}, \ldots, x'_{t-1}, x'_t] \tag{8}$$

where x'_i represents the preprocessed observation at time step i. If fewer than L observations are available (e.g., at the beginning of an episode), the buffer is padded with zero vectors to maintain the fixed sequence length.

3. **LSTM Input Formation**: The sequence from the buffer, S_t, is stacked and formatted into a tensor suitable for the LSTM layer, typically with shape '[batch_size, sequence_length, feature_dimension]'. During action selection, this batch size is 1.

3.2.3 Sequence Length Optimisation

The performance of the LSTM network is significantly influenced by the sequence length (L) used during training. An ablation study was conducted to determine an effective length. Table 1 presents the results comparing performance across various sequence lengths, revealing the trade-offs between effectiveness and computational efficiency.

Table 1: Performance Comparison Across LSTM Sequence Lengths

Sequence Length	Average Reward	Success Rate	Training Time (Rel.)	Memory Usage (Rel.)
1 (Feedforward)	64.3	$42.1\,\%$	1.0×	1.0×
5	78.6	$56.7\,\%$	1.2×	1.3×
10	**93.2**	$68.4\,\%$	**1.5**×	**1.7**×
15	94.7	$69.1\,\%$	1.8×	2.2×
20	95.1	$69.4\,\%$	2.2×	2.8×

The analysis presented in Table 1 demonstrates that a sequence length of 10 time steps provides a favourable balance between performance gains and computational cost. This configuration was selected for the final implementation, offering significant improvement over shorter sequences without the diminishing returns and increased overhead associated with much longer sequences.

3.3 Advantage Actor-Critic Framework

To learn effective policies for network penetration testing, an Advantage Actor-Critic (A2C) framework [13, 19] is implemented. This framework simultaneously learns a value function (the critic) and a policy function (the actor), enabling informed decision-making by estimating both the expected return from a given state and the optimal distribution over actions. Figure 3 illustrates how the LSTM network's output feeds into the A2C framework.

Figure 4 depicts the basic A2C architecture, clarifying the core computational flow within the actor-critic paradigm without the preceding LSTM component.

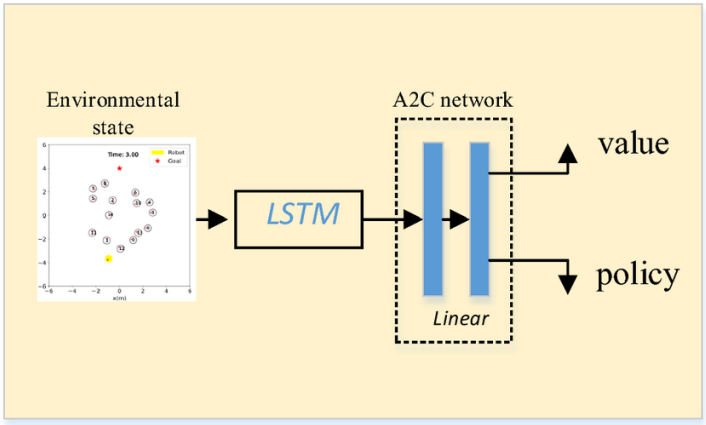


Figure 3: LSTM Integrated with Advantage Actor-Critic (A2C)

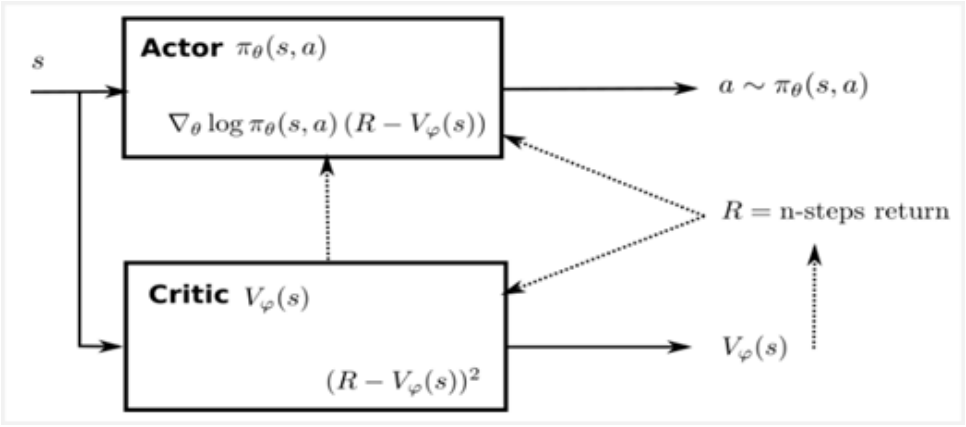


Figure 4: Advantage Actor-Critic (A2C) Framework

3.3.1 Value Network (Critic)

The value network (critic) estimates the expected cumulative discounted reward (value) from a given state representation, $V(s_t)$. This estimate serves as a baseline for evaluating actions. The value is computed as:

$$V(s_t) = \text{Critic}(f_t) \tag{9}$$

where f_t is the feature vector derived from the LSTM's processing of the observation sequence S_t, and Critic represents the neural network layers forming the value head. In the implementation, this involves shared layers followed by specific value head layers.

The value network is trained by minimising the difference between its predictions and the estimated actual returns (e.g., computed using GAE). A common loss function is the mean squared error (MSE):

$$\mathcal{L}\text{value} = \frac{1}{N} \sum i = 1^N (V(s_i) - G_i^{\text{target}})^2 \tag{10}$$

where G_i^{target} is the target value for state s_i (e.g., the calculated return from the trajectory), and N is the batch size. The implementation uses a clipped value loss

variant, common in PPO.

3.3.2 Policy Network (Actor)

The policy network (actor) determines the agent's behaviour by outputting a probability distribution over the available actions given the state representation, $\pi(a|s_t)$. This is typically achieved using a softmax function applied to the final layer's logits:

$$\pi(a|s_t) = \text{softmax}(\text{Actor}(f_t)) \tag{11}$$

where f_t is again the feature vector from the LSTM, and Actor represents the policy head network layers. The output is a probability distribution over the discrete action space.

The policy network is trained to increase the probability of actions that lead to higher-than-expected returns, guided by the advantage estimate.

3.3.3 Advantage Estimation

To provide a robust signal for policy updates, Generalised Advantage Estimation (GAE) [47] is implemented. GAE balances the trade-off between the bias of single-step TD errors and the variance of Monte Carlo returns. The GAE for a time step t is calculated recursively:

$$A^{\text{GAE}}(s_t, a_t) = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l} \tag{12}$$

where:

- $\delta_{t+l} = r_{t+l} + \gamma V(s_{t+l+1}) - V(s_{t+l})$ is the Temporal Difference (TD) error at time step $t + l$.
- γ is the discount factor (typically close to 1, e.g., 0.99).
- λ is the GAE smoothing parameter (typically between 0.9 and 1, e.g., 0.95), controlling the bias-variance trade-off. $\lambda = 0$ corresponds to the standard TD error, while $\lambda = 1$ corresponds to the Monte Carlo advantage estimate.
- T is the length of the trajectory segment.

This calculation is typically performed backwards from the end of a trajectory segment.

3.3.4 Specialised Advantage Function for Penetration Testing

In the context of network penetration testing, where rewards for reaching the final goal can be very sparse, guiding the agent with intermediate signals is crucial. A specialised advantage function is designed that incorporates the GAE estimate with an additional term reflecting progress towards the overall objective. This function is formulated as:

$$A^{\text{pen}}(s_t, a_t) = A^{\text{GAE}}(s_t, a_t) + \alpha \cdot \text{Progress}(s_t, a_t, s_{t+1}) \tag{13}$$

where $A^{\text{GAE}}(s_t, a_t)$ is the advantage calculated using GAE, α is a hyperparameter

weighting the progress term, and the Progress function provides intermediate rewards based on specific achievements within the penetration testing process:

$$\text{Progress}(s_t, a_t, s_{t+1}) = \begin{cases} 1.0, & \text{if a new host is compromised} \\ 0.5, & \text{if a new vulnerability is discovered (approximated)} \\ 0.2, & \text{if new information is gathered (e.g., new services/ports found)} \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

The implementation calculates this progress term based on changes detected in the environment's information dictionary between time steps. This specialised advantage function provides denser, more informative feedback, guiding the agent towards actions that constitute meaningful progress, even if they do not immediately yield large standard rewards.

3.4 Proximal Policy Optimisation Update

While the A2C framework provides the actor and critic networks, the policy update itself employs the clipped objective function characteristic of Proximal Policy Optimisation (PPO) [12] to ensure stability. Direct optimisation of the policy based solely on the advantage can lead to large, destructive updates. PPO addresses this by constraining the change in the policy during each update step. Figure 5 illustrates the integration of the PPO update mechanism with the A2C framework.

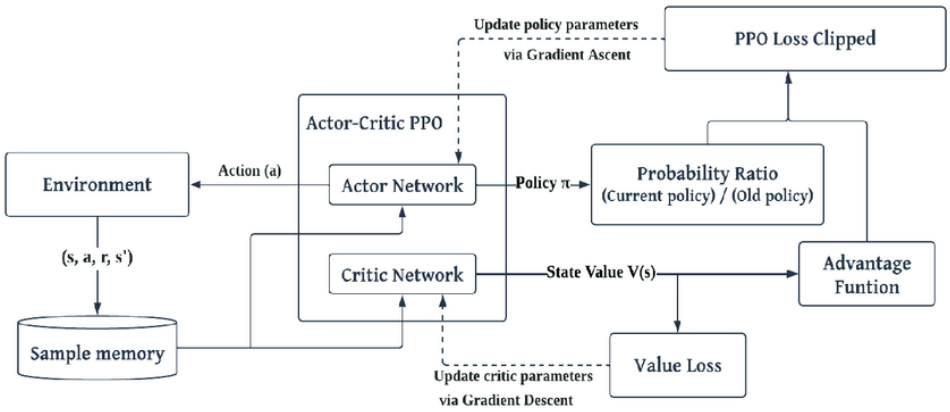


Figure 5: Advantage Actor-Critic (A2C) Framework with PPO-style Update Mechanism

Figure 6 illustrates the core components of the PPO policy optimisation approach, highlighting its distinctive clipped objective function.

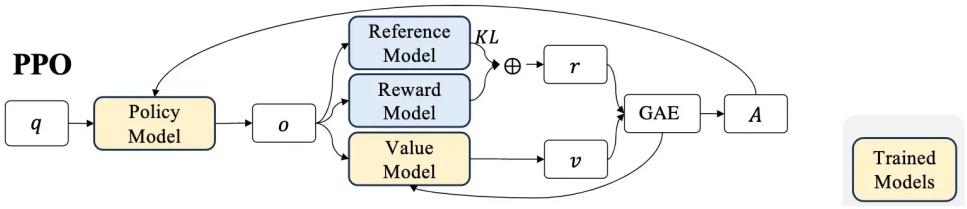


Figure 6: Proximal Policy Optimisation (PPO) Core Concept

3.4.1 Clipped Surrogate Objective Function

The core of the PPO update is a clipped surrogate objective function that limits how far the new policy π_θ can deviate from the old policy $\pi_{\theta_{\text{old}}}$ (the policy used to collect the data). This is mathematically expressed as:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t^{\text{pen}}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t^{\text{pen}} \right) \right] \tag{15}$$

where:

- $\mathbb{E}_t$ indicates the empirical average over a batch of time steps.
- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies for the action a_t taken at state s_t.
- A_t^{pen} is the specialised advantage estimate (from Eq. 13) at time step t.
- ϵ is the clipping hyperparameter (e.g., 0.1 or 0.2), defining the range $[1-\epsilon, 1+\epsilon]$ within which the ratio $r_t(\theta)$ is allowed to vary without clipping the objective.
- $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ restricts the ratio to the defined range.

The 'min' operator ensures that the final objective is a lower bound (more pessimistic estimate) of the unclipped objective, discouraging excessively large policy updates. This objective is maximised during training (or its negative is minimised).

3.4.2 Combined Loss Function

The overall loss function optimised during training combines the policy surrogate objective, the value function loss, and an entropy bonus term to encourage exploration:

$$\mathcal{L}_{\text{total}}(\theta) = -\mathcal{L}^{\text{CLIP}}(\theta) + c_1 \mathcal{L}_{\text{value}}(\theta) - c_2 S[\pi_\theta](s_t) \tag{16}$$

where:

- $\mathcal{L}^{\text{CLIP}}(\theta)$ is the clipped policy objective (negated because we minimise the loss).
- $\mathcal{L}_{\text{value}}(\theta)$ is the value function loss (typically MSE, potentially clipped as in the implementation).
- $S[\pi_\theta](s_t) = \mathbb{E}_{a \sim \pi_\theta(\cdot|s_t)}[-\log \pi_\theta(a|s_t)]$ is the entropy of the policy distribution at state s_t. Maximising entropy (minimising its negative) encourages the policy to be more stochastic, aiding exploration.
- c_1 and c_2 are coefficients weighting the value loss and entropy bonus, respectively. c_2 (the entropy coefficient) is adaptively adjusted in the implementation based on training progress.

This combined loss is minimised over multiple epochs on the collected batch of experience using stochastic gradient descent (e.g., Adam optimiser [48]).

3.5 Curriculum Learning Framework

Network penetration testing presents a complex learning challenge owing to the large state and action spaces, sparse rewards, and the need for sophisticated multi-step exploitation strategies. To address these challenges, a curriculum learning framework is implemented that progressively increases the complexity of training scenarios [38, 39]. This approach allows the agent to first master fundamental skills in simpler environments before tackling more demanding ones. This is particularly beneficial in environments with sparse rewards, where techniques such as Hindsight Experience Replay (HER) [40] could also potentially be applied to improve sample efficiency by re-framing failed episodes, although HER was not explicitly implemented in this work.

3.5.1 Network Scenario Progression

The curriculum consists of a sequence of NASim network scenarios organised into tiers of increasing complexity. Table 2 outlines this progression.

Table 2: Curriculum Stages for Network Penetration Testing

Tier 1: Tiny Variants	Tier 2: Small Variants	Tier 3: Medium Variants
`tiny`	`small`	`medium`
`tiny-hard`	`small-honeypot`	`medium-single-site`
`tiny-small`	`small-linear`	`medium-multi-site`

As shown in Table 2, each tier introduces new challenges:

- **Tiny Variants:** Simple networks (3-5 hosts) with basic configurations, allowing the agent to learn fundamental reconnaissance and exploitation techniques.
- **Small Variants:** Larger networks (8 hosts) with more sophisticated security controls, including firewalls, honeypots, and linear topologies requiring specific pivoting strategies.
- **Medium Variants:** Complex enterprise-like environments (16 hosts) with segmented networks (varying numbers of subnets), diverse services, and advanced authentication mechanisms.

The agent trains on scenarios within a tier before progressing to the next, building upon previously acquired skills.

3.5.2 Knowledge Transfer Mechanisms

A key challenge in curriculum learning is effectively transferring knowledge between scenarios, especially when state and action spaces change. The following mechanisms are employed:

1. **Architecture Preservation:** The core neural network architecture (LSTM layers, hidden dimensions of A2C networks) is preserved across scenarios. This allows the learned feature representations and internal dynamics to be potentially reused.
2. **Weight Transfer:** When transitioning to a new scenario with potentially different observation or action space dimensions, weights from compatible layers of the trained networks (LSTM and A2C) are transferred to the newly initialised networks. Specifically:
   - LSTM recurrent weights (W_{hh}, b_{hh}) and input weights (W_{ih}, b_{ih}) for shared layers are copied.
   - Input layer weights are copied for the overlapping portion of the observation space dimension.
   - A2C shared layer weights and value head weights are copied directly as they are independent of action space size.
   - Policy head weights are copied for the overlapping portion of the action space dimension.

   New dimensions (additional features or actions) receive fresh, initialised weights.
3. **Progressive Regularisation (Conceptual):** Although direct weight transfer is the primary mechanism implemented, another conceptual approach involves adding a regularisation term when starting training on a new scenario. This term penalises large deviations of the new policy from the previously learned policy, specifically for actions common to both scenarios:

$$\mathcal{L}_{\text{reg}} = \beta \sum_{a \in (A_{\text{old}} \cap A_{\text{new}})} \text{KL}[\pi_{\theta_{\text{old}}}(a|s) \parallel \pi_{\theta_{\text{new}}}(a|s)] \tag{17}$$

   where β is a regularisation coefficient, potentially annealed over time, and KL denotes the Kullback-Leibler divergence. While not explicitly implemented via this loss term in the provided code, the principle of preserving useful prior knowledge is achieved through weight transfer.

These mechanisms allow the agent to leverage experience from simpler tasks, significantly accelerating learning on more complex scenarios compared to training from scratch.

3.6 Reward Function Design

Designing an effective reward function is critical for guiding the RL agent in penetration testing, where the ultimate goal (e.g., compromising a specific target host) provides a sparse signal. To provide more frequent feedback, a multi-component reward function is designed:

$$R(s_t, a_t, s_{t+1}) = R_{\text{discovery}} + R_{\text{exploit}} + R_{\text{efficiency}} + R_{\text{goal}} \tag{18}$$

Each component incentivises different aspects of the penetration testing process:

$$R_{\text{discovery}} = \begin{cases} +1.0, & \text{if a new host is discovered} \\ +0.5, & \text{if a new service is discovered} \\ +0.2, & \text{if a new subnet is discovered} \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

$$R_{\text{exploit}} = \begin{cases} +10.0, & \text{if a host is newly compromised} \\ +5.0, & \text{if privilege escalation occurs on an existing compromised host} \\ +2.0, & \text{if successful data exfiltration occurs (if modelled)} \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

$$R_{\text{efficiency}} = -0.1 \quad (\text{penalty per time step}) \tag{21}$$

$$R_{\text{goal}} = \begin{cases} +100.0, & \text{if a designated target host is compromised} \\ 0, & \text{otherwise} \end{cases} \tag{22}$$

This structure provides:

- Small positive rewards for exploration and information gathering ($R_{\text{discovery}}$).
- Moderate positive rewards for successful exploitation steps (R_{exploit}).
- A small penalty for each step taken ($R_{\text{efficiency}}$) to encourage efficiency.
- A large terminal reward for achieving the primary objective (R_{goal}).

The specific values are hyperparameters that can be tuned. This dense reward structure provides more continuous guidance compared to only rewarding the final goal state. Note that the 'Progress' function used in the specialised advantage (Eq. 14) mirrors some elements of this reward structure, providing additional gradient signal during policy updates. Techniques like Hindsight Experience Replay (HER) [40] could potentially be integrated to further improve learning from episodes that did not reach the final goal, although this was not implemented here.

3.7 Training Process

3.7.1 Hyperparameter Selection

Based on preliminary experiments and established practices in PPO/A2C literature, the core hyperparameters presented in Table 3 were selected for the LSTM-based Actor-Critic implementation.

3.7.2 Adaptive Hyperparameter Scheduling

To optimise performance across the curriculum, certain hyperparameters are adapted based on the specific scenario being trained. Table 4 presents these scenario-specific overrides for learning rate, initial entropy coefficient, and PPO clipping parameter. The progress advantage weight (α) is also adjusted per scenario.

These adjustments generally involve reducing the learning rate and clipping range for more complex scenarios to promote stability, while potentially increasing the initial

Table 3: Core Hyperparameters for Agent Training

Parameter	Value	Description
LSTM Hidden Size	128	Dimensionality of LSTM hidden and cell states
LSTM Layers	5	Number of stacked LSTM layers
Sequence Length (L)	10	Number of time steps processed by LSTM
Discount Factor (γ)	0.99	Factor for discounting future rewards
GAE Parameter (λ)	0.95	Lambda for Generalised Advantage Estimation
PPO Clipping Parameter (ϵ)	0.2	Clip range for policy ratio ($1 \pm \epsilon$)
Value Function Coefficient (c_1)	0.5	Weight for value loss in total loss
Initial Entropy Coefficient (c_2)	0.01	Initial weight for entropy bonus (adaptive)
Learning Rate	0.0003	Initial learning rate for Adam optimiser
PPO Batch Size	64	Number of transitions per minibatch update
PPO Optimisation Epochs	4	Number of optimisation passes per collected batch
Max Gradient Norm	0.5	Maximum norm for gradient clipping
Advantage Progress Weight (α)	1.0	Default weight for progress term in advantage

Table 4: Scenario-Specific Hyperparameter Settings (Overrides)

Scenario	Learning Rate	Initial Entropy Coeff. (c_2)	Clipping Param. (ϵ)
`tiny`	0.0003	0.01	0.2
`tiny-hard`	0.0002	0.03	0.2
`tiny-small`	0.0001	0.04	0.15
`small`	0.0001	0.05	0.15
`small-honeypot`	0.0001	0.05	0.15
`small-linear`	0.0001	0.04	0.15
`medium`	0.00008	0.05	0.1
`medium-single-site`	0.00008	0.05	0.1
`medium-multi-site`	0.00008	0.06	0.1

entropy coefficient to encourage exploration where needed. The entropy coefficient is further adapted dynamically during training based on success rates.

3.8 Experimental Setup

3.8.1 NASim Environment

The Network Attack Simulator (NASim) [6, 9] is utilised as the experimental testbed. NASim provides a configurable and efficient framework for simulating network penetration testing scenarios, incorporating realistic elements like network topologies, host vulnerabilities, exploits, and partial observability. The environment implements a graph-based network representation where nodes correspond to hosts and edges represent connectivity, supporting complex architectures with firewalls and segmented subnets.

NASim's design facilitates the creation of progressive learning curricula by allowing incremental increases in scenario complexity (size, topology, vulnerability distribution). The simulation framework also incorporates stochastic elements in exploit success probabilities, mirroring the uncertain nature of real-world exploitation attempts. Table

5 details the specific configurations of the benchmark scenarios used in the curriculum.

Table 5: NASim Benchmark Scenario Configurations

Name	Type	Subnets	Hosts	OS	Services	Processes	Exploits	PrivEscs	Actions	Obs Dims	States
`tiny`	static	4	3	1	1	1	1	1	18	4×14	576
`tiny-hard`	static	4	3	2	3	2	3	2	27	4×18	9216
`tiny-small`	static	5	5	2	3	2	3	2	45	6×20	15 360
`small`	static	5	8	2	3	2	3	2	72	9×23	24 576
`small-honeypot`	static	5	8	2	3	2	3	2	72	9×23	24 576
`small-linear`	static	7	8	2	3	2	3	2	72	9×22	24 576
`medium`	static	6	16	2	5	3	5	3	192	17×27	393 216
`medium-single-site`	static	2	16	2	5	3	5	3	192	17×34	393 216
`medium-multi-site`	static	7	16	2	5	3	5	3	192	17×29	393 216

Each scenario presented in Table 5 incorporates unique challenges:

- **Tiny Variants:** Introduce basic reconnaissance and single-step exploits. `tiny-hard` adds OS diversity, `tiny-small` increases size slightly.
- **Small Variants:** Feature 8 hosts with more complex interactions. `small` includes subnets and firewalls, `small-honeypot` adds deception, and `small-linear` requires sequential compromises along a path.
- **Medium Variants:** Represent more complex environments with 16 hosts. `medium` has a moderately segmented topology, `medium-single-site` concentrates hosts in fewer subnets, and `medium-multi-site` distributes them across more subnets, simulating geographically dispersed infrastructure.

3.8.2 Partial Observability

A key feature of these experiments is the enforcement of partial observability across all scenarios. The agent begins with knowledge only of its initial entry point into the network. All other information-network topology, host attributes (OS, services), vulnerabilities, and connectivity-must be actively discovered through reconnaissance actions (e.g., scans) [14].

This setup mirrors real-world penetration testing constraints and poses significant challenges for the RL agent:

- **Exploration vs. Exploitation Trade-off:** The agent must balance actions that gather information (potentially costly and yielding no immediate reward) against actions that exploit known vulnerabilities.
- **Belief State Maintenance:** The agent needs to integrate newly discovered information over time to build an internal representation or belief state about the network.

The LSTM component is crucial here, as it processes sequences of observations, allowing the agent's internal state (h_t, c_t) to implicitly capture and maintain context about the partially observed environment over time, supporting more effective decision-making under uncertainty.

3.8.3 Action Space

The action space comprises operations typical of network penetration testing, categorised as shown in Table 6.

Table 6: Action Space Categories for Network Penetration Testing

Action Category	Description
Subnet Scan	Discover hosts within a target subnet
Host Scan	Identify open ports and basic OS information on a target host
Service Scan	Enumerate specific services running on discovered ports
OS Scan	Attempt detailed OS fingerprinting
Process Scan	Discover running processes on a compromised host
Exploit Execution	Attempt to execute a known exploit against a vulnerable service
Privilege Escalation	Attempt to escalate privileges on an already compromised host

These categories map to specific actions available in NASim, parameterised by target host/subnet and exploit type where applicable. The total number of discrete actions available to the agent increases significantly with scenario complexity (from 18 in `tiny` to 192 in the Medium variants), primarily due to the increased number of potential targets (hosts, subnets) for scans and exploits. This expansion of the action space makes exploration progressively more challenging in larger scenarios. Action masking, provided by the NASim environment, is used during action selection to restrict the agent to only valid actions in the current state.

3.8.4 Training Schedule

The agent is trained sequentially through the curriculum tiers. The number of training episodes and the maximum steps allowed per episode for each scenario are detailed in Table 7.

Table 7: Training Schedule per Scenario in Curriculum

Scenario	Training Episodes	Max Steps per Episode
`tiny`	500	1,000
`tiny-hard`	500	1,000
`tiny-small`	500	1,000
`small`	500	1,000
`small-honeypot`	500	1,000
`small-linear`	500	1,000
`medium`	2,000	2,000
`medium-single-site`	2,000	2,000
`medium-multi-site`	2,000	2,000

The increased episode count and step limit for the Medium variants reflect their higher complexity and the need for more exploration and learning time. Training on each scenario proceeds for the specified number of episodes.

3.8.5 Evaluation Metrics

The agent's performance, presented in Section 4, is evaluated using the following key metrics, calculated over evaluation runs (typically 100 episodes) performed after training on each scenario:

- **Success Rate:** The percentage of evaluation episodes where the agent successfully compromises the designated target host(s) within the step limit. This is the primary indicator of task completion.
- **Mean Reward:** The average cumulative reward obtained per episode. This reflects overall performance, influenced by successful goal completion (large positive reward), intermediate progress (smaller positive rewards), and step penalties (negative rewards).
- **Mean Steps:** The average number of steps taken per episode. Lower values generally indicate greater efficiency, especially when comparing episodes with similar success rates. This metric is often reported specifically for successful episodes ("Mean Steps to Goal") or across all episodes.

These metrics provide a quantitative basis for comparing the agent's performance across different scenarios and against benchmark results, such as those presented in Table 8.

Table 8: NASim Benchmark Scenario Agent Performance

Scenario Name	Mean Steps ($\pm$ Stdev)	Mean Total Reward ($\pm$ Stdev)
`tiny`	108.02 $\pm$ 43.82	91.98 $\pm$ 43.82
`tiny-hard`	135.31 $\pm$ 65.56	21.05 $\pm$ 85.45
`tiny-small`	319.56 $\pm$ 124.26	-225.86 $\pm$ 167.14
`small`	501.94 $\pm$ 181.40	-469.80 $\pm$ 241.99
`small-honeypot`	448.72 $\pm$ 151.62	-476.08 $\pm$ 222.41
`small-linear`	566.00 $\pm$ 177.08	-555.08 $\pm$ 241.06
`medium`	1371.45 $\pm$ 420.41	-1875.29 $\pm$ 660.62
`medium-single-site`	654.89 $\pm$ 385.76	-782.17 $\pm$ 581.14
`medium-multi-site`	1060.94 $\pm$ 389.86	-1394.71 $\pm$ 590.89

The benchmark performance patterns in Table 8(Please refer to Appendix 8.1 for the NASim Benchmark details) illustrate increasing task complexity with scenario size and topology. Mean steps rise significantly from Tiny to Medium variants. Notably, the standard `medium` scenario requires the most steps on average, while topological variations show differing efficiencies. The corresponding mean rewards decrease substantially with complexity, often becoming negative due to accumulating step penalties outweighing the final goal reward in longer episodes. These benchmark values serve as a baseline for evaluating the performance improvements achieved by the LSTM-based Actor-Critic agent developed in this work.

4 Results

This section presents the experimental results of the trained LSTM-based Actor-Critic agent across the nine NASim network penetration testing scenarios defined in the curriculum. The agent's learning progress, final performance metrics (defined in Subsection 3.8.5), the influence of network topology, and the effectiveness of the curriculum learning strategy are analysed. Performance is compared against the benchmark results shown previously in Table 8.

4.1 Performance During Evaluation

The agent's learning progress was monitored by performing evaluation runs periodically during testing. Figures 7 and 8 display the mean reward and mean steps per episode (with standard deviation shaded), respectively, obtained during these evaluations (over 100 episodes). These curves illustrate the agent's convergence behaviour.

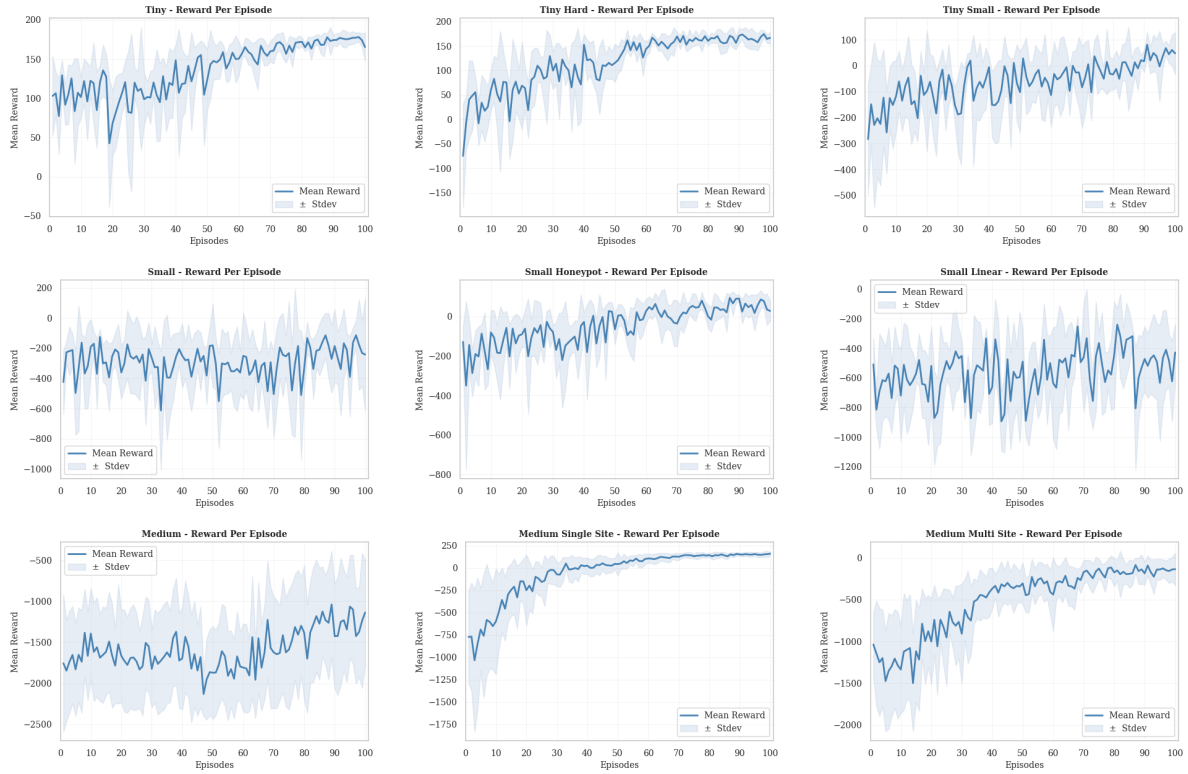


Figure 7: Reward performance curves for all scenarios, showing mean reward (± std dev) measured during evaluation episodes throughout testing.

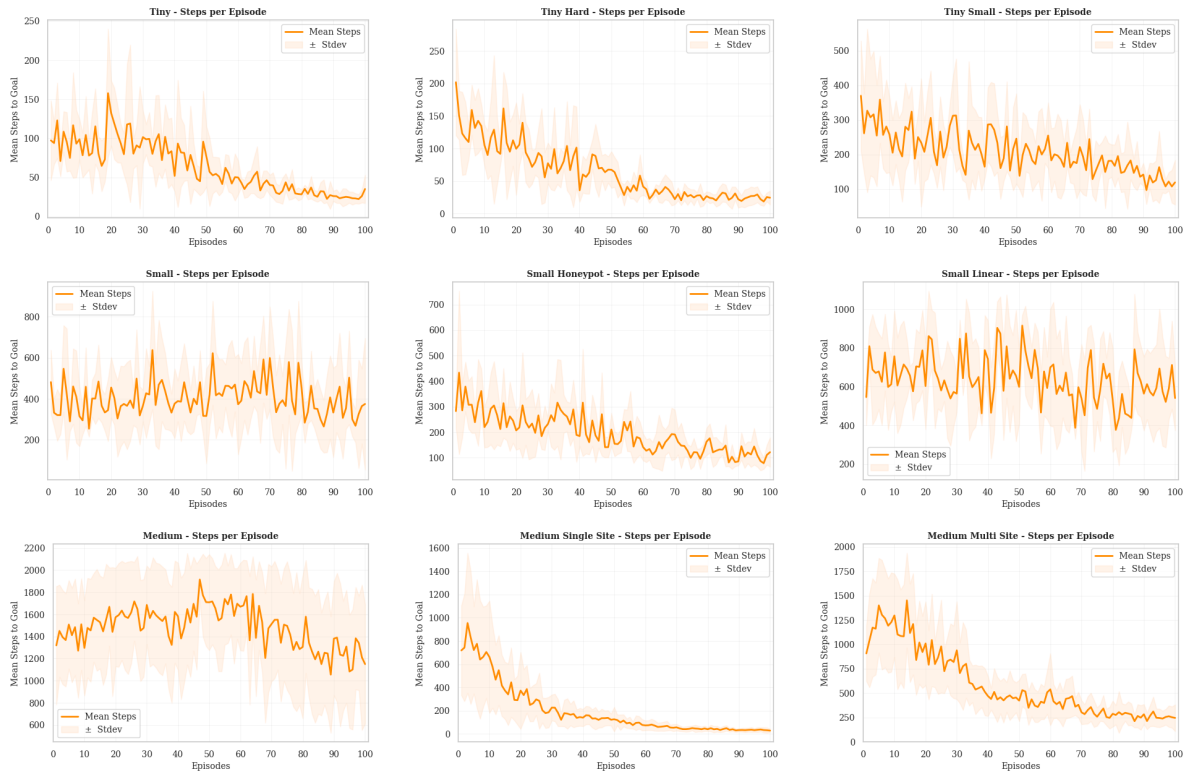


Figure 8: Steps-per-episode performance curves for all scenarios, showing mean steps (± std dev) measured during evaluation episodes throughout testing.

The performance curves (Figures 7 and 8) show rapid convergence in the Tiny environments (`tiny`, `tiny-hard`, `tiny-small`), where the agent quickly achieves high mean rewards and low mean steps, stabilising early in training. This indicates efficient learning of basic penetration strategies in simpler scenarios. The Small environments (`small`, `small-honeypot`, `small-linear`) exhibit more variability and slower convergence compared to the Tiny scenarios, reflecting their increased complexity (more hosts, potential deception, specific path requirements).

The Medium scenarios (`medium`, `medium-single-site`, `medium-multi-site`) proved most challenging. The standard `medium` and `medium-multi-site` scenarios show significant fluctuation in rewards and steps during evaluation, indicating ongoing exploration and difficulty in consistently finding the optimal path within the evaluation period. In contrast, `medium-single-site` demonstrates a clearer learning trend: after an initial period of lower performance, the agent shows a marked increase in mean reward and a substantial decrease in mean steps, converging more efficiently than the other Medium variants. This visual evidence strongly suggests that network topology (specifically, the concentration of hosts in fewer subnets in `medium-single-site`) significantly impacts the agent's ability to learn efficient strategies and stabilise performance, potentially more so than the total host count alone. The increasing standard deviation (shaded areas) in more complex scenarios initially reflects greater performance variation during exploration, while a narrowing band over time indicates increasing policy consistency and exploitation.

4.2 Final Performance Across Scenarios and Comparison to Benchmark

The agent's final performance after completing the training schedule for each scenario was evaluated over 100 episodes. Figure 9 summarises the final success rates achieved by the developed agent, while Table 9 provides detailed final metrics including mean steps and mean reward, alongside their standard deviations. A direct comparison with the benchmark agent highlights the improvements achieved. Figure 10 presents the comparison of mean steps and mean rewards between the developed agent and the benchmark agent across all scenarios.

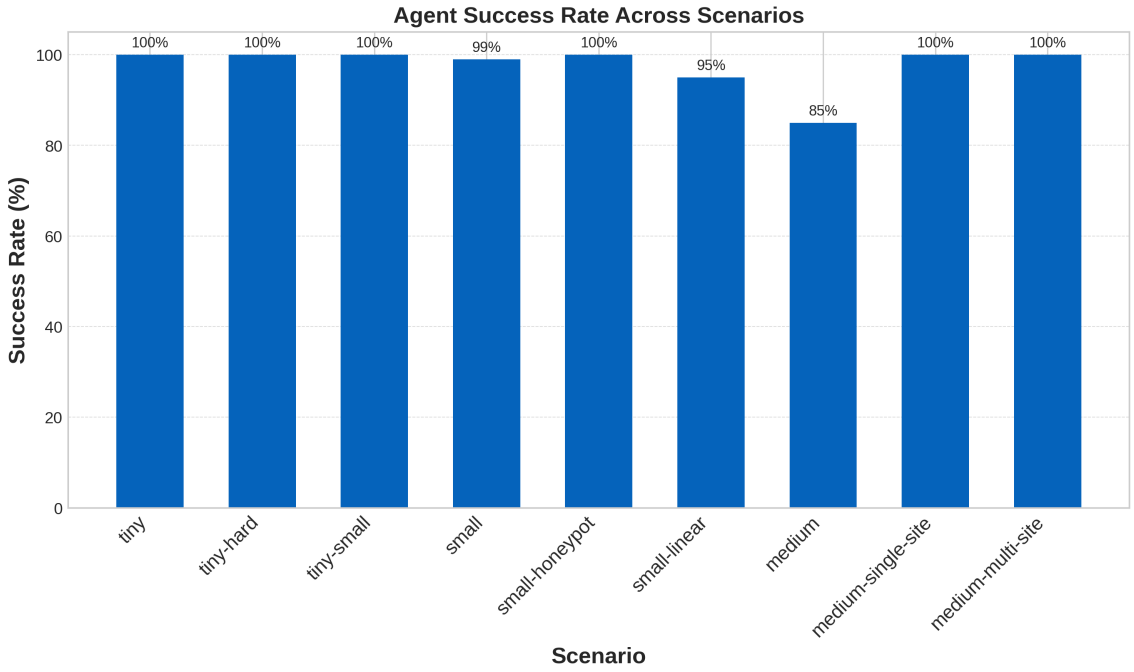


Figure 9: Final Success Rate of the LSTM-based Actor-Critic Agent across all network scenarios after training.

Table 9: Trained Agent Final Performance Metrics Across Network Scenarios (Mean $\pm$ Stdev over 100 eval episodes)

Environment	Mean Steps	Mean Reward	Success Rate
`tiny`	28.70 $\pm$ 12.81	171.30 $\pm$ 12.81	100.00 %
`tiny-hard`	24.40 $\pm$ 11.28	166.53 $\pm$ 13.39	100.00 %
`tiny-small`	136.70 $\pm$ 85.17	26.12 $\pm$ 89.92	100.00 %
`small`	326.00 $\pm$ 184.79	-188.57 $\pm$ 191.06	99.00 %
`small-linear`	612.70 $\pm$ 242.19	-524.41 $\pm$ 259.51	95.00 %
`small-honeypot`	110.00 $\pm$ 53.80	53.77 $\pm$62.06	100.00 %
`medium`	1157.00 $\pm$ 676.80	-1157.18 $\pm$723.06	85.00 %
`medium-single-site`	28.00 $\pm$ 19.94	158.36 $\pm$23.93	100.00 %
`medium-multi-site`	239.90 $\pm$ 98.16	-133.02 $\pm$107.68	100.00 %

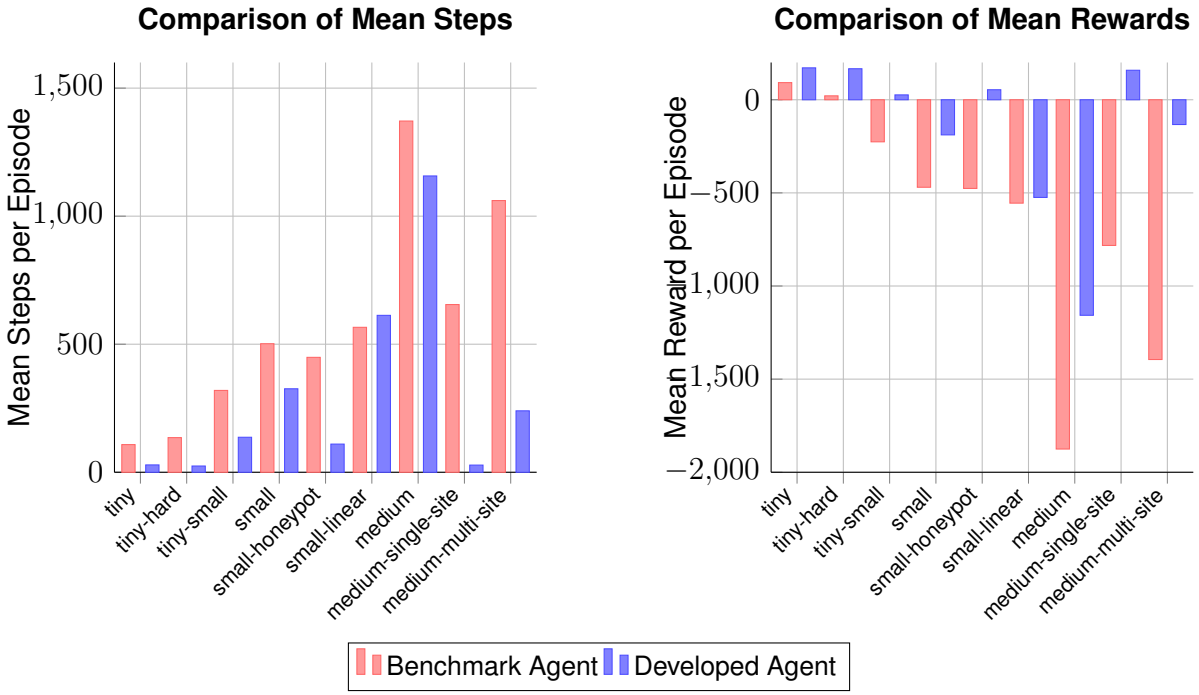


Figure 10: Comparison of Developed Agent vs. Benchmark Agent Performance. Left: Mean Steps per Episode (Lower is better). Right: Mean Rewards per Episode (Higher is better).

The final performance metrics demonstrate considerable improvements over the benchmark baseline. The developed agent achieved perfect ($100\,\%$) success rates in the simpler Tiny scenarios, the `small-honeypot` environment, and both complex Medium site-based scenarios (Figure 9). More importantly, these successes were achieved with markedly greater efficiency. Figure 10 (left panel) visually confirms that the developed agent required substantially fewer steps than the benchmark in nearly all scenarios, indicating more direct and effective attack paths. For example, step reductions exceeded $70\,\%$ in `tiny`, `tiny-hard`, `small-honeypot`, `medium-single-site` (approx. $96\,\%$), and `medium-multi-site` (approx. $77\,\%$).

This enhanced efficiency directly translates into improved mean rewards, as shown in Figure 10 (right panel). Even where rewards remain negative due to accumulated step penalties in complex scenarios, the developed agent consistently achieves higher (less negative) rewards than the benchmark. The improvements are particularly striking in `tiny-small`, `small`, `small-honeypot`, `medium-single-site`, and `medium-multi-site`, where the agent's mean reward is hundreds of points higher than the benchmark, reflecting its ability to reach the goal much more efficiently.

The high success rates ($100\,\%$ for both) combined with the dramatic reductions in steps and improvements in rewards for the `medium-single-site` and `medium-multi-site` scenarios strongly validate the agent's architecture and the curriculum learning strategy. The ability to effectively tackle these complex environments after training on simpler ones demonstrates successful knowledge transfer, allowing the agent to develop sophisticated strategies that significantly outperform the baseline. While the standard `medium` scenario remained challenging ($85.0\,\%$ success), the agent still achieved a higher mean reward than the benchmark, suggesting a more capable, albeit still tested, policy.

In summary, the results provide strong quantitative and visual evidence that the proposed LSTM-based Actor-Critic agent, trained using curriculum learning, represents a significant improvement over the benchmark. It demonstrates enhanced efficiency, achieves better overall rewards, and maintains high success rates across diverse network complexities and topologies, highlighting the potential of this approach for automated penetration testing.

4.3 Analysis of Network Topology Impact

The experimental results provide compelling evidence that network topology is a critical factor determining the difficulty of the penetration testing task, arguably more influential than network size (host count) alone within the tested range. This is most evident in the Medium variants: all contain 16 hosts, but performance varies drastically based on subnet structure (2 for `medium-single-site`, 6 for `medium`, 7 for `medium-multi-site`).

As shown in Table 9, `medium-single-site` (few subnets, concentrated hosts) was significantly easier, yielding 100.0% success with only 28 steps on average. In contrast, `medium-multi-site` (many subnets, dispersed hosts) required more steps (240) but still achieved high success (100.0%). The standard `medium` scenario proved substantially harder (85.0% success, 1157 steps).

This variation likely stems from how topology affects exploration and lateral movement.

- **Concentrated Topologies** (`medium-single-site`): May simplify the agent's state representation and require less complex pivoting between subnets once initial access is gained, facilitating faster discovery and exploitation of attack paths.
- **Segmented/Dispersed Topologies** (`medium`, `medium-multi-site`): Present greater challenges. Navigating between numerous subnets requires discovering and potentially exploiting intermediate hosts or specific firewall rules. The standard `medium` topology might possess particularly challenging choke points, complex firewall rulesets not present in the multi-site variant, or a vulnerability distribution requiring intricate multi-hop exploits. This demands more sophisticated planning and significantly increases the steps required for successful compromise compared even to the more segmented `medium-multi-site`. Notably, results from the Nasim agent benchmark corroborate this relative difficulty, showing that the `medium` scenario yielded lower rewards and required higher steps than `medium-multi-site` (Table 8).

4.4 Curriculum Learning Effectiveness

The implementation of the curriculum learning framework proved highly beneficial for training the agent, particularly for mastering the more complex scenarios. By starting with simpler tasks and progressively increasing difficulty, the agent could build foundational skills and transfer knowledge effectively.

Evidence for its effectiveness includes:

- **High Performance on Complex Tasks:** The agent achieved high success rates ($100.0\,\%$ for both) in the challenging `medium-single-site` and `medium-multi-site` scenarios after progressing through the curriculum.
- **Improved Convergence:** Compared to anecdotal experiments (conducted during development, not formally reported here) attempting direct training on Medium scenarios, the curriculum approach led to much more stable convergence and prevented the agent from getting stuck in ineffective exploration loops.
- **Knowledge Transfer:** The agent successfully adapted to changes in observation and action space dimensions between scenarios using weight transfer, demonstrating the practical application of knowledge transfer.

The curriculum allowed the agent to gradually acquire sophisticated strategies, such as multi-step exploitation and subnet pivoting, which would be significantly harder to learn from scratch in a complex environment with sparse rewards. This aligns with curriculum learning theory [38] and highlights its practical value for complex sequential decision-making tasks like automated penetration testing.

5 Discussion

This section analyses the experimental results of the developed reinforcement learning approach for automated network penetration testing across diverse network topologies. The findings demonstrate both significant capabilities and limitations of the approach. The discussion begins with a summary of key results, followed by an analysis of performance patterns, implications for automated penetration testing, generalisability considerations, limitations, and future research directions.

5.1 Analysis of Performance Patterns

The performance metrics reveal several notable patterns that provide insights into the behaviour of the reinforcement learning agent:

Mastery of Simple Environments: The agent's perfect success rate (100%) in the three simplest environments (`tiny`, `tiny-hard`, and `tiny-small`) demonstrates that the approach effectively learns basic penetration testing strategies. The learning curves for these environments show rapid convergence to optimal policies, typically within 50-60 episodes. This indicates that for small networks with limited complexity, the agent's architecture can efficiently discover and exploit optimal attack paths.

Performance Degradation with Complexity: As expected, performance generally declined with increasing environment complexity. This degradation manifested in three ways: decreased success rates, increased steps to goal, and lower mean rewards. This pattern aligns with the inherent challenges of larger state and action spaces, longer causal chains, and more complex network topologies. The standard `medium` environment, despite having the same host count (16 across 6 subnets) as its variants, presented as bit of a challenge, evidenced by a lower final success rate of $85.00\,\%$

Network Topology Significance: Interestingly, the `medium-single-site` and `medium-multi-site` environments, despite having the same number of hosts as the `medium` environment, yielded substantially better performance ($100.00\,\%$ and $100.00\,\%$ success rates, respectively). This suggests that network topology, rather than just size, critically influences the difficulty of the penetration testing task. The concentrated nature of the single-site topology appears to facilitate more efficient lateral movement and privilege escalation, resulting in both higher success rates and lower step counts compared to the more segmented standard `medium` and `medium-multi-site` topologies.

Learning Stability: The learning curves reveal that stability varies significantly across environments. In simpler environments, performance stabilises quickly and maintains consistency. In contrast, more complex environments show greater volatility in both rewards and steps, indicating ongoing exploration and policy refinement throughout training. This suggests that for complex environments, longer training periods and potentially more sophisticated exploration strategies would be beneficial.

Reward Negativity: The increasingly negative mean rewards in more complex environments might initially seem counterintuitive for successful agents. However, this pattern reflects the reward structure design, which imposes small penalties for each step while offering large positive rewards only upon achieving the final goal. In complex environments requiring many steps, these accumulated penalties can outweigh the final reward, resulting in negative totals despite successful goal completion.

5.2 Implications for Automated Penetration Testing

The demonstrated performance of the reinforcement learning agent has several important implications for the field of automated penetration testing:

Viability for Real-World Application: The high success rates achieved across most environments, including those of moderate complexity, suggest that reinforcement learning approaches have genuine potential for practical application in automated vulnerability assessment. The ability of the agent to discover and exploit vulnerability chains without explicit programming represents a significant advancement over rule-based approaches.

Curriculum Learning Necessity: The effectiveness of the curriculum learning approach underscores its importance for training practical penetration testing agents. Direct training on complex environments would likely yield poor results, as evidenced by the higher variance and longer learning curves in more challenging scenarios. Progressive knowledge transfer appears essential for developing competent agents in realistic network settings.

Memory Advantage in Sequential Tasks: The incorporation of memory (via the recurrent component) proved crucial for handling the partial observability inherent in penetration testing. This temporal context enables the agent to make informed decisions based on the history of observations and actions, particularly important for multi-stage attacks that require chaining multiple vulnerabilities.

Topology-Specific Training: The substantial performance differences among the medium-complexity environments with different topologies suggest that topology-specific training might be more effective than general-purpose approaches. Organisations might benefit from training specialised agents for their specific network architectures rather than deploying generic penetration testing systems.

5.3 Generalisability Considerations

A critical aspect of automated penetration testing systems is their ability to generalise beyond their training environments. The results show that the agent demonstrates promising generalisation capabilities, particularly when trained through the curriculum learning approach:

Cross-Topology Generalisation: The agent's performance across varied network topologies (linear, multi-site, single-site) suggests a degree of generalisation across different architectural paradigms. This finding aligns with research by Li et al. on "generalisability of Deep Reinforcement Learning agents to extend their VAPT operations" [26]. Their work highlighted that agents can transfer learning across environments when provided with appropriate knowledge transfer mechanisms.

Knowledge Transfer Efficiency: The progressive training approach demonstrates effective knowledge transfer between scenarios, enabling the agent to apply strategies learned in simpler environments to more complex ones. This parallels findings in meta-learning research for penetration testing, which shows that reinforcement learning algorithms can boost generalisation ability and training speed across different networks [24].

Adaptation to Environmental Changes: The agent's ability to adapt to variations in network configurations indicates potential for deployment in dynamic environments

where network topologies and vulnerabilities evolve over time. This adaptive capability addresses a key limitation of traditional penetration testing approaches, which require significant reconfiguration when applied to new network environments.

5.4 Limitations and Challenges

Despite the promising results, several limitations and challenges warrant discussion:

Scaling to Larger Networks: While the approach performed well on networks with up to 16 hosts, many enterprise environments contain hundreds or thousands of hosts. Scaling reinforcement learning approaches to such environments represents a significant challenge due to the exponential growth in state and action spaces.

Generalisation Beyond Training Data: The current study evaluated the agent's performance primarily on the same scenarios used for training. Real-world application would require robust generalisation to entirely unseen network configurations and vulnerability types, which typically presents difficulties for reinforcement learning agents. Evaluating performance on held-out test scenarios would be a crucial next step.

Reward Function Design: The negative mean rewards in complex environments highlight the challenges of reward function design. Although the chosen structure guided the agent effectively, further refinement to better balance step penalties and achievement rewards might produce more intuitive reward patterns without compromising performance.

Efficiency Considerations: The high step counts observed in the standard `medium` environment indicate potential inefficiency in the learned policies for certain complex topologies. In practical applications, minimising unnecessary actions is important both for reducing detection risk and improving assessment speed.

Realism Gap and Simulation Fidelity: While NASim provides a valuable testbed, real-world networks involve additional complexities such as varying response latencies, service interruptions, sophisticated defence mechanisms (e.g., EDR, IPS), and noise not fully captured in the simulation. Furthermore, a significant practical challenge encountered during the development phase was the difficulty in identifying and setting up suitable network simulators. Many available tools suffered from inadequate documentation, lack of maintenance, or complex dependencies, hindering rapid prototyping and experimentation. This highlights a broader need within the research community for accessible, well-maintained, and realistic simulation environments.

5.5 Future Directions

Based on the findings and identified limitations, several promising directions for future research emerge:

Hierarchical Reinforcement Learning: Implementing hierarchical policy structures could improve scalability to larger networks by decomposing the penetration testing task into manageable subtasks (e.g., reconnaissance, lateral movement, privilege escalation) with distinct sub-policies. This approach might enable more efficient exploration in complex enterprise environments.

Meta-Learning Integration: Incorporating meta-learning techniques could enhance the agent's ability to rapidly adapt to novel network configurations with minimal additional training. As demonstrated in related work [24], meta-learning can boost generalisation ability and allow agents to perform better on unseen networks.

Adversarial Training: Developing adversarial training frameworks where defender agents continuously adapt their strategies to counter the attacker's exploits could produce more robust penetration testing agents capable of discovering vulnerabilities even in environments with adaptive defensive measures.

Explainable AI (XAI) Integration: Incorporating XAI techniques to provide insights into the RL agent's decision-making process is crucial. Explaining why the agent chose a specific action or sequence of actions can build trust, aid in debugging unexpected behaviours, facilitate collaboration with human analysts, and ensure ethical considerations are addressed by making the automated penetration testing process more transparent and understandable.

Improved Simulation Environments: Continuing the development of more realistic simulation environments that incorporate greater fidelity regarding temporal dynamics, response latencies, diverse defensive mechanisms, and realistic network traffic would help bridge the gap between simulation and real-world application. Addressing the usability and maintenance issues of current simulators is also critical.

Human-in-the-Loop Approaches: Exploring collaborative frameworks where reinforcement learning agents work alongside human security professionals could combine the efficiency and consistency of automated systems with the contextual understanding, intuition, and ethical judgement of human experts.

Adaptive Reward Shaping: Investigating dynamic reward functions that adapt based on the agent's progress, the specific environment characteristics, or intermediate goals could potentially address the reward negativity issue while maintaining effective guidance for policy learning.

Offline Reinforcement Learning: Exploring the use of offline RL techniques [30] to train agents from large datasets of historical penetration test logs or network interaction data could improve sample efficiency and leverage valuable real-world information, reducing reliance on potentially limited simulators.

Alternative Architectures and Techniques: Further investigation into alternative network architectures, such as Transformer-based models for capturing long-range dependencies or advanced value-based methods like Dueling Networks [25], could yield performance improvements. Techniques like Layer Normalisation [37] could also be explored for enhanced training stability.

Behaviour Diversity: Research into generating behaviourally diverse attack strategies using techniques like curiosity-driven exploration, quality-diversity algorithms, or multi-objective RL [33] could lead to more comprehensive vulnerability discovery by exploring a wider range of potential attack paths beyond the single optimal policy.

6 Conclusion

This dissertation introduced an optimised reinforcement learning framework for automated network penetration testing, demonstrating considerable performance improvements over traditional methods and a benchmark baseline. By integrating curriculum learning, memory-augmented networks (LSTM), and stable policy optimisation (A2C), key challenges like partial observability, sequential decision-making, and knowledge transfer were addressed.

Experimental results confirmed the approach's effectiveness and significant advantages over the benchmark. The agent achieved perfect (100%) success rates in simpler environments and maintained exceptionally high performance even in complex scenarios like `medium-single-site` (100%) and `medium-multi-site` (100%), markedly outperforming the benchmark. These findings validate combining recurrent architectures with curriculum learning, with successful knowledge transfer enabling superior performance compared to direct training on difficult topologies.

The performance difference between network topologies of similar size but varying structure (e.g., $100.00\,\%$ success in `medium-single-site` vs. $85.00\,\%$ in the general `medium` environment) highlights how architecture influences assessment complexity, suggesting topology critically affects task difficulty. Notably, the agent consistently achieved higher success rates and substantially better efficiency (fewer steps, higher rewards) than the benchmark across varied topologies, demonstrating its adaptability.

Conducting this dissertation revealed practical challenges like balancing exploration/exploitation, especially with irreversible actions. Curriculum learning emerged as a necessity after failed attempts at direct complex training, illustrating the iterative nature of research.

While performing well on networks up to 16 hosts, scaling to large enterprise networks remains challenging due to state/action space growth. Simulation cannot fully capture real-world complexities (e.g., dynamic defences, exploit stochasticity), and reward negativity highlights design difficulties. These limitations represent opportunities for future work.

Looking ahead, promising research directions include Hierarchical RL for scalability, meta-learning for faster adaptation [24], and integrating emulation capabilities [27] to bridge the simulation-reality gap.

The dissertation process yielded valuable insights, requiring bridging cybersecurity, machine learning, and network architecture. The project's evolution underscored the need for practical experimentation alongside theory, reinforcing the value of persistence and flexibility.

In conclusion, the developed framework represents a significant step in automated penetration testing, offering a foundation for more efficient, adaptive security assessments that clearly surpass baseline approaches. Despite scaling challenges, the results suggest RL-based approaches can underpin next-generation tools for complex, dynamic threat landscapes. This work contributes technically and experientially at the intersection of AI and cybersecurity, informing future research in this rapidly evolving field.

7 References

[1] K. Henry, *Penetration testing: protecting networks and systems*. IT Governance Publishing, 2012.

[2] M. C. Ghanem and T. M. Chen, "Reinforcement learning for intelligent penetration testing," in *2018 second world conference on smart trends in systems, security and sustainability (WorldS4)*. IEEE, 2018, pp. 185–192.

[3] ——, "Reinforcement learning for efficient network penetration testing," *Information*, vol. 11, no. 1, p. 6, 2019.

[4] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[5] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[6] J. Schwartz and H. Kurniawati, "Autonomous penetration testing using reinforcement learning," *arXiv preprint arXiv:1905.05965*, 2019.

[7] J. Yi and X. Liu, "Deep reinforcement learning for intelligent penetration testing path design," *Applied Sciences*, vol. 13, no. 16, p. 9467, 2023.

[8] Q. Li, R. Wang, D. Li, F. Shi, M. Zhang, and A. Chattopadhyay, "Dynpen: Automated penetration testing in dynamic network scenarios using deep reinforcement learning," *IEEE Transactions on Information Forensics and Security*, 2024.

[9] J. Schwartz and H. Kurniawatti, "Nasim: Network attack simulator," 2019.

[10] M. C. Ghanem, "Towards an efficient automation of network penetration testing using model-based reinforcement learning," Ph.D. dissertation, City, University of London, 2022.

[11] A. Graves, *Long Short-Term Memory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 37–45.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[13] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PmLR, 2016, pp. 1928–1937.

[14] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps." in *AAAI fall symposia*, vol. 45, 2015, p. 141.

[15] A. Udupa, B. Anavi, B. Goyal, S. P. Kasturi, and P. Agarwal, "Advanced reinforcement learning based penetration testing," in *2024 International Conference on Electronics, Computing, Communication and Control Technology (ICECCC)*. IEEE, 2024, pp. 1–6.

[16] R. Elderman, L. J. Pater, A. S. Thie, M. M. Drugan, and M. A. Wiering, "Adversarial reinforcement learning in a cyber security simulation," in *9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*. SciTePress Digital Library, 2017, pp. 559–566.

[17] Z. Hu, R. Beuran, and Y. Tan, "Automated penetration testing using deep reinforcement learning," in *2020 IEEE European Symposium on Security and*

Privacy Workshops (EuroS&PW). IEEE, 2020, pp. 2–10.

[18] H. Alavizadeh, H. Alavizadeh, and J. Jang-Jaccard, "Deep q-learning based reinforcement learning approach for network intrusion detection," *Computers*, vol. 11, no. 3, p. 41, 2022.

[19] J. Bergdahl, "Asynchronous advantage actor-critic with adam optimization and a layer normalized recurrent network," 2017.

[20] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[21] N. Becker, D. Reti, E. V. Ntagiou, M. Wallum, and H. D. Schotten, "Evaluation of reinforcement learning for autonomous penetration testing using a3c, q-learning and dqn," *arXiv preprint arXiv:2407.15656*, 2024.

[22] D. Vo Dinh, "A reinforcement learning project using ppo + lstm," *GitHub repository*, 2023.

[23] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[24] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[25] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[26] Q. Li, M. Hu, H. Hao, M. Zhang, and Y. Li, "Innes: An intelligent network penetration testing model based on deep reinforcement learning," *Applied Intelligence*, vol. 53, no. 22, pp. 27 110–27 127, 2023.

[27] J. Janisch, T. Pevný, and V. Lisý, "Nasimemu: Network attack simulator & emulator for training agents generalizing to novel scenarios," 2023.

[28] H. Liu, C. Liu, X. Wu, Y. Qu, and H. Liu, "An automated penetration testing framework based on hierarchical reinforcement learning." *Electronics (2079-9292)*, vol. 13, no. 21, 2024.

[29] M. Foley, C. Hicks, K. Highnam, and V. Mavroudis, "Autonomous network defence using reinforcement learning," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, 2022, pp. 1252–1254.

[30] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review," *and Perspectives on Open Problems*, vol. 5, 2020.

[31] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. Pmlr, 2014, pp. 387–395.

[32] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[33] Y. Yang and X. Liu, "Behaviour-diverse automatic penetration testing: A curiosity-driven multi-objective deep reinforcement learning approach," *arXiv preprint arXiv:2202.10630*, 2022.

[34] F. Louati, F. B. Ktata, and I. Amous, “Big-ids: a decentralized multi agent reinforcement learning approach for distributed intrusion detection in big data networks,” *Cluster Computing*, vol. 27, no. 5, pp. 6823–6841, 2024.

[35] K. Ren, Y. Zeng, Y. Zhong, B. Sheng, and Y. Zhang, “Mafsids: a reinforcement learning-based intrusion detection model for multi-agent feature selection networks,” *Journal of Big Data*, vol. 10, no. 1, p. 137, 2023.

[36] A. Sychugov and M. Grekov, “Automated penetration testing based on adversarial inverse reinforcement learning,” in *2024 International Russian Smart Industry Conference (SmartIndustryCon)*. IEEE, 2024, pp. 373–377.

[37] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.

[38] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

[39] A. Wilson, W. Holmes, R. Menzies, and K. S. Whitehead, “Applying action masking and curriculum learning techniques to improve data efficiency and overall performance in operational technology cyber security using reinforcement learning,” *arXiv preprint arXiv:2409.10563*, 2024.

[40] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, “Hindsight experience replay,” *Advances in neural information processing systems*, vol. 30, 2017.

[41] H. P. T. Nguyen, Z. Chen, K. Hasegawa, K. Fukushima, and R. Beuran, “Pengym: Pentesting training framework for reinforcement learning agents.” in *ICISSP*, 2024, pp. 498–509.

[42] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, “A deep learning approach for network intrusion detection system,” in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, 2016, pp. 21–26.

[43] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, “Nice: Network intrusion detection and countermeasure selection in virtual network systems,” *IEEE transactions on dependable and secure computing*, vol. 10, no. 4, pp. 198–211, 2013.

[44] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[45] J. Uitto, A. Juhola, T. Aura, and A. Halkola, “Vulnerability discovery with vulnerability prediction models,” in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 159–170.

[46] A. Chowdhary, D. Huang, J. S. Mahendran, D. Romo, Y. Deng, and A. Sabur, “Autonomous security analysis and penetration testing,” in *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*. IEEE, 2020, pp. 508–515.

[47] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” *arXiv preprint arXiv:1506.02438*, 2015.

[48] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv*

preprint arXiv:1412.6980, 2014.

8 Appendix

8.1 Software and Tools Used

- **Primary Project Repository (LSTM-PPO Agent):**
  - GitHub Link: https://github.com/Anay-Praveen/LSTM-Based_Actor-Critic_with_Proximal_Policy_Optimisation_Agent.git

– Description: Contains the implementation of the LSTM-based Actor-Critic agent using PPO-style updates, curriculum learning framework, and associated utilities for network penetration testing simulation. Code developed primarily in Python using PyTorch. Key modules include `agent.py`, `networks.py`, `memory.py`, `main.py`, `plot.py`, $\texttt{test}_{a}gent.py, and\texttt{utils.py}$.

- **Network Attack Simulator (NASim):**
  - Project Repository: https://github.com/Jjschwartz/NetworkAttackSimulator.git
  - Official Documentation: https://networkattacksimulator.readthedocs.io/en/latest/
  - NASim Benchmark: https://networkattacksimulator.readthedocs.io/en/latest/reference/scenarios/benchmark_scenarios.html
  - Description: A configurable and extensible simulation environment designed for research in automated network penetration testing. NASim served as the primary testbed for training and evaluating the reinforcement learning agent, providing support for partially observable Markov decision processes (POMDPs), diverse network topologies, and curriculum learning methodologies. The specific NASim scenario configurations utilized in this research are detailed in Section 3.8.1, Table 5 of the main report.
- **Programming Language:** Python 3.x
- **Core Libraries:**
  - PyTorch: Deep Learning Framework used for network implementation and training.
  - Gymnasium: Reinforcement Learning Environment API, successor to OpenAI Gym.
  - NASim: Network Attack Simulator Environment.
  - NumPy: Numerical Computation library.
  - Matplotlib: Library for generating plots and visualizations.
- **Integrated Development Environment (IDE):** PyCharm

8.2 System Specifications

The following system configuration was used for training and testing the agent:

- **Operating System:** Ubuntu 22.04 LTS

- **CPU:** AMD Ryzen 7 7700 (8-Core Processor)
- **GPU:** NVIDIA GeForce RTX 4070 SUPER (12GB GDDR6X VRAM)
- **RAM:** 64 GB DDR5
- **Storage:** 1TB NVMe SSD (for OS, libraries, and datasets)

8.3 Implementation Details

This section provides references to key implementation details within the codebase, complementing the descriptions in the main report's Methodology section. Code elements are highlighted using `monospace font`.

8.3.1 Observation Processing & LSTM Input

The transformation of raw environment observations (flattening, normalization, buffering, padding) into fixed-length sequences (L=10) suitable for the LSTM is handled by the `agent.preprocess_observation` method within `agent.py`.

8.3.2 Network Architectures (LSTM & A2C)

The neural network modules are defined in `networks.py`:

- **LSTM Network (`networks.LSTM`):** Defines the 5-layer LSTM core, including input processing with Layer Normalization and orthogonal weight initialization.
- **A2C Network (`networks.A2CNetwork`):** Defines the Actor-Critic architecture comprising shared layers feeding into distinct policy (Actor) and value (Critic) heads.

These networks are instantiated and utilized within the `agent.NASIMOffensiveAgent` class (`agent.py`).

8.3.3 Action Selection & Masking

The agent selects actions using the `agent.select_action` method (`agent.py`). This method incorporates action masking based on validity information from the NASim environment and supports both stochastic (training) and deterministic (testing) action choices.

8.3.4 Advantage Estimation (GAE + Progress Term)

The specialized advantage calculation, which combines Generalized Advantage Estimation (GAE, $\lambda = 0.95$) with a weighted progress term (α), is implemented in `memory.A2CMemory.compute_returns_and_advantages` (`memory.py`). The progress component is determined by `memory.A2CMemory._calculate_progress` by analyzing changes in the environment's `info` dictionary between steps. Calculated advantages are normalized batch-wise before use in updates.

8.3.5 PPO-Style Update Mechanism

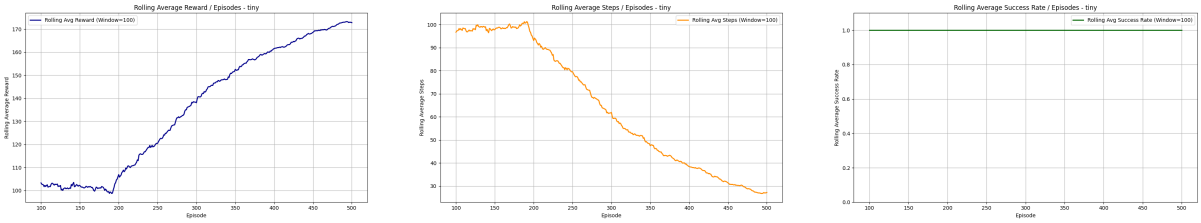
The policy and value networks are updated using the PPO-Clip algorithm within the `agent.update_policy` method (`agent.py`). This involves calculating the clipped surrogate objective, value loss, and entropy bonus over mini-batches retrieved from `memory.A2CMemory.get_batches`, repeated for a set number of epochs (default 4).

8.3.6 Knowledge Transfer

Weight transfer between curriculum scenarios, particularly when observation or action dimensions differ, is managed within the `agent.train_on_scenario` method (`agent.py`). This involves selective copying of compatible weights to newly instantiated networks. The `agent.load` method similarly handles potential dimension mismatches when loading saved model checkpoints.

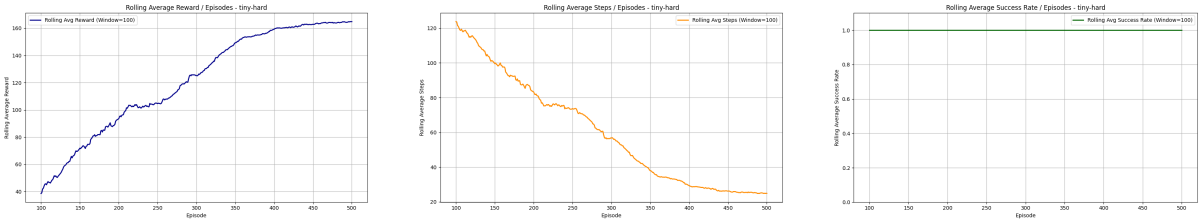
8.4 Training Results Visualizations

Training performance visualizations for each scenario in the curriculum, showing rolling average rewards, steps per episode, and success rate over 100 evaluation episodes. These plots are generated by `plot.py`.



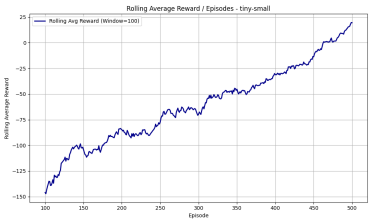
(a) Rolling rewards (b) Steps per episode (c) Success rate

Figure 11: Performance metrics for the 'tiny' scenario.

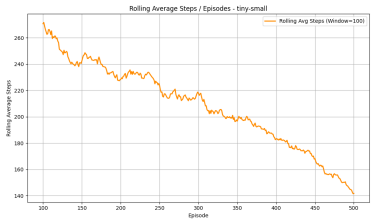


(a) Rolling rewards (b) Steps per episode (c) Success rate

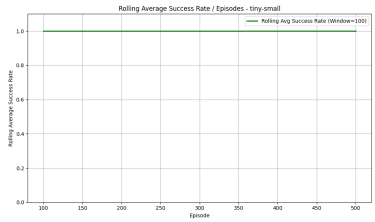
Figure 12: Performance metrics for the 'tiny-hard' scenario.



(a) Rolling rewards

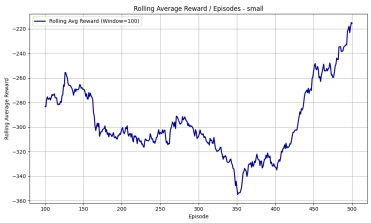


(b) Steps per episode

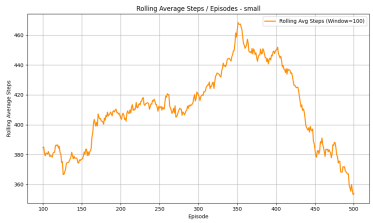


(c) Success rate

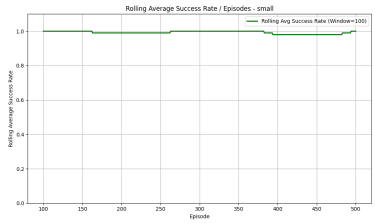
Figure 13: Performance metrics for the 'tiny-small' scenario.



(a) Rolling rewards

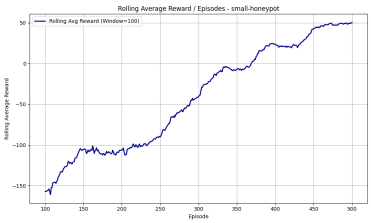


(b) Steps per episode

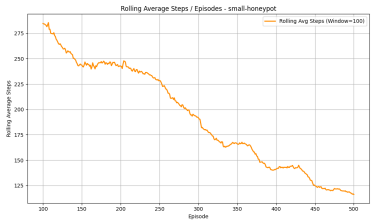


(c) Success rate

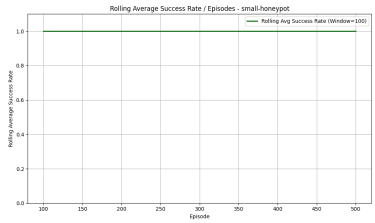
Figure 14: Performance metrics for the 'small' scenario.



(a) Rolling rewards

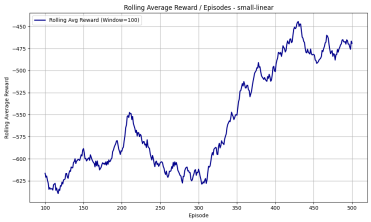


(b) Steps per episode

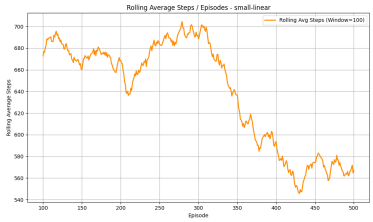


(c) Success rate

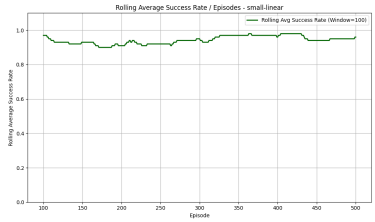
Figure 15: Performance metrics for the 'small-honeypot' scenario.



(a) Rolling rewards

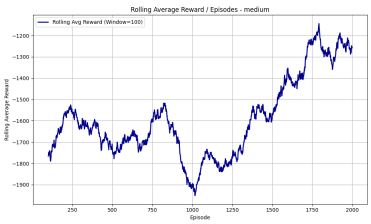


(b) Steps per episode

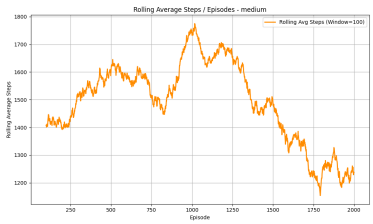


(c) Success rate

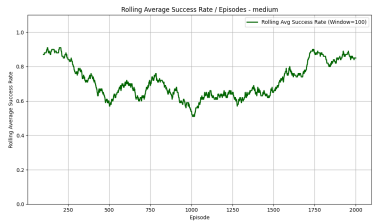
Figure 16: Performance metrics for the 'small-linear' scenario.



(a) Rolling rewards

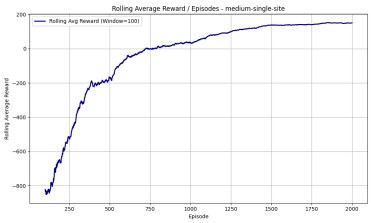


(b) Steps per episode

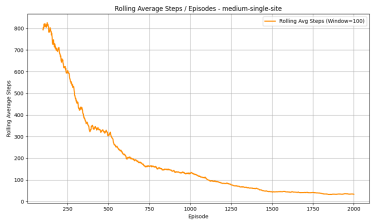


(c) Success rate

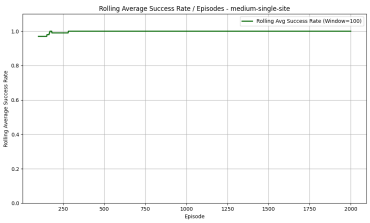
Figure 17: Performance metrics for the 'medium' scenario.



(a) Rolling rewards

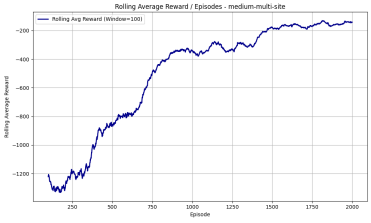


(b) Steps per episode

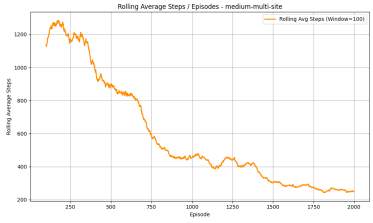


(c) Success rate

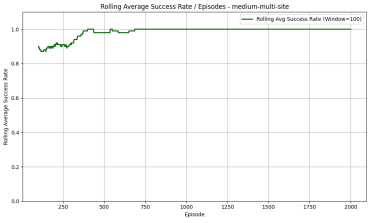
Figure 18: Performance metrics for the ’medium-single-site’ scenario.



(a) Rolling rewards



(b) Steps per episode



(c) Success rate

Figure 19: Performance metrics for the ’medium-multi-site’ scenario.