EPPTA: Efficient Partially Observable Reinforcement Learning Agent for Penetration testing Applications

Zegang Li[1], Qian Zhang[2], and Guangwen Yang[1]

[1]Tsinghua University
[2]Zhengzhou University

September 7, 2023

Abstract

In recent years, penetration testing (pen-testing) has emerged as a crucial process for evaluating the security level of network infrastructures by simulating real-world cyber-attacks. Automating pen-testing through reinforcement learning (RL) facilitates more frequent assessments, minimizes human effort, and enhances scalability. However, real-world pen-testing tasks often involve incomplete knowledge of the target network system. Effectively managing the intrinsic uncertainties via partially observable Markov decision processes (POMDPs) constitutes a persistent challenge within the realm of pen-testing. Furthermore, RL agents are compelled to formulate intricate strategies to contend with the challenges posed by partially observable environments, thereby engendering augmented computational and temporal expenditures. To address these issues, this study introduces EPPTA (Efficient POMDP-Driven Penetration Testing Agent), an agent built on an asynchronous RL framework, designed for conducting pen-testing tasks within partially observable environments. We incorporate an implicit belief module in EPPTA, grounded on the belief update formula of the traditional POMDP model, which represents the agent's probabilistic estimation of the current environment state. Furthermore, by integrating the algorithm with the high-performance RL framework, Sample Factory, EPPTA significantly reduces convergence time compared to existing pen-testing methods, resulting in an approximately 20-fold acceleration. Empirical results across various pen-testing scenarios validate EPPTA's superior task reward performance and enhanced scalability, providing substantial support for efficient and advanced evaluation of network infrastructure security.

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

EPPTA: Efficient Partially Observable Reinforcement Learning Agent for Penetration testing Applications

Zegang Li[1,2] | Qian Zhang[2,3] | Guangwen Yang*[1,2]

[1]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[2]National Supercomputing Center in Wuxi, Wuxi, China
[3]School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou, China

Correspondence
*Guangwen Yang,
Tsinghua University, Beijing, China.
Email: ygw@tsinghua.edu.cn

Abstract

In recent years, penetration testing (pen-testing) has emerged as a crucial process for evaluating the security level of network infrastructures by simulating real-world cyber-attacks. Automating pen-testing through reinforcement learning (RL) facilitates more frequent assessments, minimizes human effort, and enhances scalability. However, real-world pen-testing tasks often involve incomplete knowledge of the target network system. Effectively managing the intrinsic uncertainties via partially observable Markov decision processes (POMDPs) constitutes a persistent challenge within the realm of pen-testing. Furthermore, RL agents are compelled to formulate intricate strategies to contend with the challenges posed by partially observable environments, thereby engendering augmented computational and temporal expenditures. To address these issues, this study introduces EPPTA (**E**fficient **P**OMDP-Driven **P**enetration **T**esting **A**gent), an agent built on an asynchronous RL framework, designed for conducting pen-testing tasks within partially observable environments. We incorporate an implicit belief module in EPPTA, grounded on the belief update formula of the traditional POMDP model, which represents the agent's probabilistic estimation of the current environment state. Furthermore, by integrating the algorithm with the high-performance RL framework, Sample Factory, EPPTA significantly reduces convergence time compared to existing pen-testing methods, resulting in an approximately 20-fold acceleration. Empirical results across various pen-testing scenarios validate EPPTA's superior task reward performance and enhanced scalability, providing substantial support for efficient and advanced evaluation of network infrastructure security.

KEYWORDS:
Penetration testing, asynchronous RL, partial observable, optimizations

1 | INTRODUCTION

Penetration testing has become a fundamental element in comprehensive network security strategies, assessing the security of computer systems, networks, and applications by simulating malicious actor-driven attacks. In this context, the automation of pen-testing has gained significant importance. It aims to alleviate the time and cost limitations associated with traditional manual methods and adapt to the complexities of modern networks[1]. However, this shift toward automation is accompanied by notable challenges, especially within intricate pen-testing environments.

In recent years, reinforcement learning has found extensive application in autonomous decision-making challenges. Within the realm of pen-testing, it exhibits remarkable potential. RL is capable of facilitating agent learning and decision-making by optimizing reward signals, offering potential cost-effectiveness, rapid prototype design, secure testing environments, and reproducibility in the domain of automated pen-testing[2]. Nonetheless, in practical pen-testing scenarios, agents frequently confront the constraints of partial observability, a challenge accentuated within the framework of RL-based pen-testing automation. Attackers often possess access only to limited and potentially unreliable information sources, such as network traffic and system logs, thereby restricting comprehensive access to the target system.

Partial observable reinforcement learning offers an avenue to address the challenges posed by incomplete information environments, with POMDPs serving as a widely adopted framework for their resolution. Nevertheless, effective and efficient implementation of RL-based automated pen-testing techniques in the context of partial observability remains an unresolved challenge in the pen-testing field. Despite diverse methodologies proposed for tackling POMDPs, such as the fusion of model-free RL with recurrent models[3,4], or approximative techniques employing neural networks and traditional POMDP models[5,6], the effective integration of these methods to tackle the intricate environments within pen-testing warrants further investigation.

To address this research gap, this study introduces an intelligent agent named "Efficient POMDP-Driven Penetration Testing Agent" (EPPTA), leverages a high-performance RL framework to tackle pen-testing tasks within partially observable environments. EPPTA's design draws inspiration from advanced methodologies in asynchronous RL and partial observability, aiming to overcome challenges faced by conventional methods in pen-testing.

One fundamental innovation in our proposed approach lies in the incorporation of an implicit belief module, drawing inspiration from the belief update formulation found in traditional POMDP theory. The belief module equips EPPTA with the capacity to make precise estimations of the probability distribution regarding the states of the environment based on the partial observations it receives. Within the belief module, we have employed long short-term memory (LSTM) layers[7] to store trajectory observation data and select optimal actions guided by our belief module. This enhancement substantially augments EPPTA's decision-making capacity, particularly in pen-testing scenarios characterized by partial observability, enabling EPPTA to devise more effective attack sequence strategies informed by the agent's beliefs.

Moreover, we seamlessly integrated the EPPTA algorithm into a high-performance asynchronous RL framework known as "Sample Factory"[8]. This framework empowers EPPTA to asynchronously perform data collection and parameter gradient updates across multiple parallel pen-testing environments. By leveraging shared memory mechanisms, we achieved a significant reduction in data transfer overhead between parallel environments. Additionally, we adopted a double-buffered sampling approach to ensure continuous acquisition of experiential data by the agent, thereby optimizing its learning process. This integration has resulted in a substantial enhancement in the convergence efficiency of EPPTA and has equipped the algorithm with improved scalability.

To validate the effectiveness of EPPTA, we conducted experiments in the simulated pen-testing environment NASim[9]. Our results indicate that EPPTA successfully and efficiently exploits vulnerabilities within partially observable pen-testing scenarios, manifesting as a reduction in both the number of attack steps and convergence time. As an illustrative example, consider POCP2Gen, one of the largest default environments in NASim, featuring over 3500 actions and a state space of $\sim 2^{27}$. Within POCP2Gen, EPPTA not only converges to an average number of attack steps closer to theoretical values but also exhibits convergence speeds approximately 20 times faster than existing algorithms.

Furthermore, when confronting compromised network systems, human experts frequently employ node isolation as a defense mechanism. To replicate such scenarios, we introduced controlled random node isolation in our experiments, effectively preventing the agent from accessing information related to the isolated nodes. Notably, our approach consistently outperforms traditional methods in these scenarios, demanding fewer average attack steps and less time consumption to access the target host's information.

In conclusion, our research contributes to the development of efficient autonomous cybersecurity systems by enhancing the efficiency and effectiveness of pen-testing. This advancement holds the potential to assist organizations in safeguarding their systems and networks from cyber threats, ensuring the confidentiality, integrity, and availability of sensitive data and critical infrastructure.

2 | RELATED WORKS

Autonomous pen-testing: During the nascent stage of research, pen-testing planning employed attack graphs and decision trees to depict a variety of attack actions through the conjunctive combination of pre-conditions and post-conditions pertaining to the relevant properties of network systems[10]. These methodologies were closely linked to traditional planning techniques and aimed to discover the most effective attack graph. While these algorithms produce interpretable and formal models, they are predominantly appropriate for small-to-medium-sized networks, facing considerable obstacles when attempting to scale to larger and more intricate networks[11]. In a recent study by Gangupantulu et al.[12], RL was incorporated with attack graphs to enhance the scalability of such models. However, this strategy necessitates the acquisition of prior network information and the deployment of tailor-made MDPs.

RL in pen-testing: In recent times, there has been a considerable increase in research efforts employing RL algorithms to address pen-testing challenges[2,13,14]. These problems have been modeled as fully observable MDPs. A number of novel architectures, incorporating domain-specific modifications for deep RL, have been developed, including the double agent architecture[15], a hierarchical decomposition method termed HA-DQN[16], and various improvements to the DQN algorithm with the Intrinsic Curiosity Module (ICM) called NDSPI-DQN[17]. Concurrently, Hu et al.[18] successfully identified efficient attack paths for exploitation from a range of possible solutions generated by traditional search algorithms, specifically targeting vulnerability assessments within particular subnets.

RL for POMDPs: Numerous RL approaches have been developed to address the complexities inherent in POMDPs, concentrating on belief update models and recurrent models. Belief update-centric methodologies, including Point-Based Value Iteration (PBVI)[19] and Monte Carlo Value Iteration (MCVI)[20], approximate the belief state by managing a collection of belief points and updating them during the learning process. These strategies have been successful in scaling to larger POMDPs with discrete state and action spaces. On the other hand, recurrent models utilize memory-based structures to maintain information about the environment over time. LSTM networks and Gated Recurrent Units (GRUs)[21] have been integrated into deep RL algorithms to enable the learning of policies in partially observable settings. In this vein, model-free RL methods such as Deep Recurrent Q-Networks (DRQN)[22] have been proposed, that demonstrated the potential of recurrent model-free RL in addressing POMDPs. Recently, researchers have also explored the development of specialized belief modules[23] and sequential model learning architectures[24] to facilitate convergence in specific problem domains. These approaches aim to improve RL's effectiveness in handling POMDPs by incorporating domain-specific modifications and adaptive learning mechanisms.

High-performance RL frameworks: High-performance RL frameworks have become increasingly important in addressing the computational challenges associated with POMDPs. POMDPs introduce additional complexities due to the uncertainty in the environment, which necessitates the development of efficient and scalable training methods for RL agents. One notable framework in this category is the Asynchronous Advantage Actor-Critic (A3C)[25] and its generalized counterpart (GA3C)[26]. These frameworks are specifically tailored to the parallelized training of RL agents, offering particular advantages in POMDPs. The concurrent interaction of multiple agents with the environment is particularly beneficial for exploration in such scenarios. Another significant contribution to this area is the Importance Weighted Actor-Learner Architecture (IMPALA)[27], is a scalable and distributed RL framework designed to handle POMDPs efficiently. IMPALA employs off-policy correction methods V-trace and decouples acting and learning processes, making it suitable for large-scale distributed training. Sample Factory, an advanced high-performance RL framework, takes the spotlight by prioritizing both single-node sample efficiency and computational scalability[8]. It utilizes shared memory and shared GPU memory technologies to improve internal communication efficiency and maximize the computing capabilities of a single node. In addition, Sample Factory's adaptive frame-skipping technique dynamically modifies the number of frames omitted during training, enabling agents to focus on critical observations within partially observable contexts. Furthermore, several high-performance RL libraries, including Stable Baselines[28] and RLlib[29], have made significant contributions to the field. These libraries offer a unified framework, streamline hyperparameter tuning, and enable distributed training, thereby advancing the state of high performance RL. Their effectiveness in tackling the computational challenges inherent in complex RL tasks has been well-documented, resulting in improved scalability and training efficiency. By leveraging these advanced frameworks and libraries, increasingly sophisticated RL agents can be developed to address uncertainty and partial observability effectively, thereby contributing to the cutting-edge advancements in POMDP-based RL.

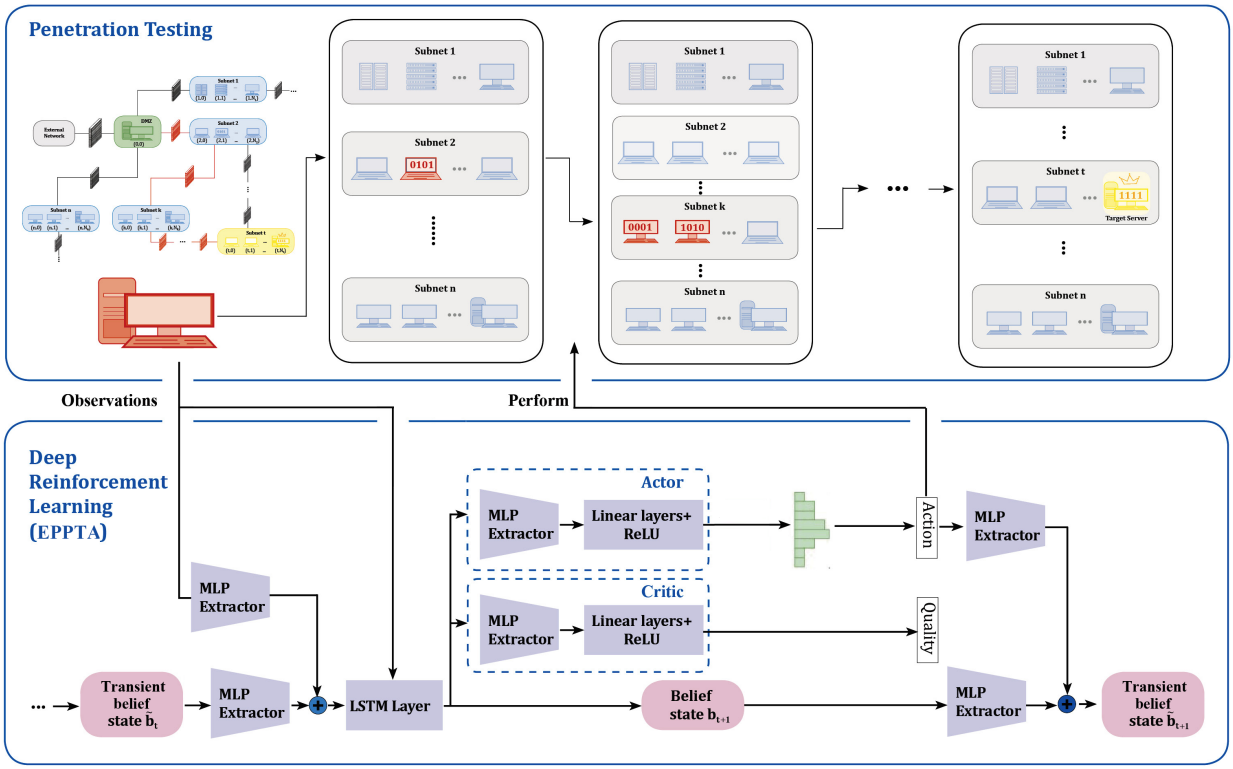

Figure 1 A schematic representation of the proposed EPPTA method is provided. The upper portion illustrates an outline of the partially observable pen-testing process, while the lower portion details the EPPTA flowchart. An innovative belief module is introduced, which collaborates with the Actor-Critic network framework to implicitly comprehend the belief update for the pen-testing environment state.

3 | PRELIMINARIES

3.1 | Pen-testing

Pen-testing typically involves a series of operations that adhere to a similar pattern. The process commences with an entry point, serving as a foothold for an initial attack. The initially compromised hosts may not possess adequate privileges, posing challenges for attackers in further network penetration. The ultimate objective of an attack is to compromise the target machine and maintain influence over it, necessitating lateral movement within the network to acquire higher privileges in the system. More specifically, the environment for conducting pen-testing usually comprises multiple subnets, each containing several hosts, such as servers or terminal devices, as depicted in the upper portion of Figure 1. The pen-testing process is generally initiated by an attacker situated in the demilitarized zone (DMZ), who begins by utilizing techniques such as port and vulnerability scanning to collect information about the target system, network, or application and identify any known vulnerabilities or weaknesses. Following this, the attacker strives to access hosts within adjacent subnets in the network topology, potentially employing social engineering tactics or exploiting known vulnerabilities to gain entry. Once access is obtained, the attacker may seek to escalate the privileges of the compromised host using techniques such as password cracking, using it as a starting point for subsequent testing phases. This iterative process continues until the attacker successfully accesses the final target host, represented by the golden host, and exploits its vulnerabilities. Ultimately, the attacker furnishes a detailed record of the sequence of attack actions, as depicted by the red lines in the figure, for further investigation. Reducing the number of attack steps (indicated by shorter red lines) provides defenders with less time to detect critical security vulnerabilities within the network system. Therefore, the number of attack steps is a crucial metric for evaluating network security.

3.2 | POMDPs for pen-testing

3.2.1 | Modeling

Pen-testing presents a unique challenge due to its partially observable nature, where attackers must make decisions based on limited information. POMDPs offer a suitable framework for modeling such scenarios in which an agent must make decisions under uncertainty.

In the context of pen-testing, a POMDP can be defined by a tuple $(S, A, P, R, \Omega, O, \gamma)$, where:

- S represents the set of possible system states, including various network configurations, access privileges, and security measures. Each element $s \in S$ denotes a distinct system state, with its values representing information such as penetrated network topology, firewalls, and access privilege levels.
- A denotes the set of actions that an attacker can perform, where $a \in A$ denotes a specific attack action. The Adversarial Tactics Techniques and Common Knowledge (ATT&CK) framework[30] serves as an example of the action set, which includes a variety of operations, such as exploits, service scanning from one host to another, and subnet IP scanning from a host to a subnet.
- P:$S \times A \to \Delta(S)$ is the state transition probability function, $P(s'|s, a)$, which captures the probability of transitioning to state s' upon executing action a in state s. Δ symbolizes the probability simplex.
- R:$S \times A \to \mathbb{R}$ is the reward function, $R(s, a, s')$, which quantifies the desirability of taking action a in state s and reaching state s'. In pen-testing, rewards may be associated with successful exploits, privilege escalation, or acquiring valuable information.
- Ω denotes the set of observations perceivable by the attacker, where $\omega \in \Omega$ represents the explicitly obtained observations during a pen-testing simulation. Observations are contingent upon the current system state s and the action a executed. The attacker can acquire the observable state of the subnet or host that is currently under attack or has been compromised, incorporating information such as responses from port scanning and results from vulnerability scanning.
- O:$S \times A \to \Delta(\Omega)$ is the observation probability function, $O(o|s, a)$, which models the likelihood of perceiving observation o after taking action a in state s.
- $\gamma \in [0, 1]$ is the discount factor that determines the importance of immediate versus future rewards.

3.2.2 | Belief Update

Belief update is a critical technique for estimating the current pen-testing system state from a sequence of observations and actions. Due to the inherent uncertainty of system states in POMDPs, an attacker cannot directly observe the true state of the target network. Instead, they must rely on a probability distribution over states, referred to as the belief state.

Mathematically, the belief state $b(s)$ represents a probability distribution over the set of possible states S, where $s \in S$. The belief state can be denoted as $b_t(s_t) = P(s_t \mid h_t)$, with h_t signifying the history of actions and observations up to timestep t. The belief state is updated according to the underlying POMDP model, the actions taken a_t, and the observations received ω_t. The objective is to compute the posterior probability $b_{t+1}(s_{t+1}) = P(s_{t+1} \mid h_t, a_{t+1}, \omega_{t+1})$. To establish a stronger connection between our proposed EPPTA neural network framework and the belief update formula, we will deconstruct and provide clarity on the belief update formula below.

Applying Bayes' theorem:

$$P(s_{t+1} \mid h_t, a_{t+1}, \omega_{t+1}) = \frac{P(\omega_{t+1} \mid s_{t+1}, h_t, a_{t+1}) P(s_t \mid h_t, a_{t+1})}{P(\omega_{t+1} \mid h_t, a_{t+1})}. \tag{1}$$

Notice that ω_{t+1} is conditionally independent of h_t, given s_{t+1} and a_{t+1}. Consequently, $P(\omega_{t+1} \mid s_{t+1}, h_t, a_{t+1}) = P(\omega_{t+1} \mid s_{t+1}, a_{t+1})$, which corresponds to the observation probability function $O(\omega_{t+1} \mid s_{t+1}, a_{t+1})$ previously mentioned.

Subsequently, the law of total probability is employed to determine $P(s_{t+1} \mid h_t, a_{t+1})$:

$$P(s_{t+1} \mid h_t, a_{t+1}) = \sum_{s_t \in S} P(s_{t+1} \mid s_t, h_t, a_{t+1}) P(s_t \mid h_t). \tag{2}$$

Due to the Markov property in POMDP models, we can simplify the term $P(s_{t+1} \mid s_t, h_t, a_{t+1})$ to $P(s_{t+1} \mid s_t, a_{t+1})$. Substituting back into (1):

$$b_{t+1}(s_{t+1}) = \frac{O(\omega_{t+1} \mid s_{t+1}, a_{t+1}) \sum_{s_t \in S} P(s_{t+1} \mid s_t, a_{t+1}) b_t(s_t)}{P(\omega_{t+1} \mid a_{t+1}, b_t)}. \tag{3}$$

By iteratively applying the belief update formula, an attacker can maintain an estimate of the system state, which informs their decision-making process.

3.3 | Proximal Policy Optimization (PPO)

The proposed EPPTA framework is designed to be compatible with most on-policy actor-critic reinforcement learning (RL) algorithms. In our research, we adopt the on-policy actor-critic method known as Proximal Policy Optimization (PPO)[31] for training the agent. PPO exhibits excellent adaptability to environmental changes, making it a stable choice for RL tasks where the agent's policy needs frequent updates.

The objective function of PPO is defined as follows:

$$L^{\text{PPO}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(\text{clip} \left(p_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t, p_t(\theta) \hat{A}_t \right) \right], \tag{4}$$

where $L^{\text{PPO}}(\theta)$ is the PPO loss function parameterized by θ. The term $p_t(\theta)$ represents the probability ratio between the old policy and the new policy, which quantifies the change between the previous and current policy distributions. To further enhance the efficiency of training, we utilize the Generalized Advantage Estimator (GAE)[32] to estimate the advantage function $\hat{A}_t$. The GAE computation is as follows:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma \lambda)^l \delta_{t+l}^V, \tag{5}$$

where the discount factor γ accounts for the relative importance of future rewards, while λ controls the trade-off between bias and variance in the advantage estimation. Notably, δ_{t+l}^V represents the advantage prediction error at time step $t + l$ with respect to the value function V, measuring the alignment between expected values and observed outcomes. As such, δ_{t+l}^V plays a pivotal role in the RL training process by guiding policy updates based on the quality of predictions relative to actual experiences.

4 | METHODOLOGIES

In this section, we present our approach to addressing two primary challenges in automated pen-testing: 1) the limited observability of scenarios and 2) the substantial training time overhead. To tackle these challenges, we propose an innovative and efficient RL framework named EPPTA. Figure 1 provides an overview of the EPPTA algorithm.

EPPTA incorporates multilayer perceptrons (MLPs) for feature extraction, LSTM layers to handle sparse rewards, and adopts the actor-critic framework. Within the EPPTA framework, we introduce a novel belief module along with the associated belief function loss. Transient belief states are integrated into the actor-critic framework, facilitating implicit belief state updates. This innovation promotes enhanced interaction between the agent's neural network and the environment, leveraging the belief update formula. Furthermore, we accelerate the proposed EPPTA based on the Sample Factory framework. We perform asynchronous computations for critical RL components, utilize GPU shared memory for efficient storage of belief states, and employ the double-buffered sampling strategy to seamlessly integrate EPPTA with Sample Factory. The training procedure of our method is outlined in Algorithm 1 and will be further detailed in the subsequent subsections.

4.1 | Belief Module

The belief module is a critical component of our proposed EPPTA framework, designed to efficiently capture the belief state of the pen-testing agent amid the uncertainties and partial observability inherent in the environment.

Given that the cardinality of the set S, denoted as |S|, exhibits exponential growth to an immense quantity, we introduce modifications to the conventional belief state distribution $b_t(s_t)$. We employ a matrix b_t of the same dimensions as the state matrix s to represent the current belief state, as illustrated in the subsequent equation:

Algorithm 1 Asynchronous Training of EPPTA with Parallel Environments

Input: Initialize all networks and parameters

while not converged **do**
  while environments not terminated **do**
    Asynchronously update transient belief $\tilde{b}_t$ for each environment
    for each parallel environment i in parallel **do**
      Encode $\tilde{b}_{t,i}$ and observation ω_t to estimate belief state $b_{t+1,i}$
      Execute action $a_{t,i}$ following policy π_i, obtain reward r_t, and new observation $\omega_{t+1,i}$
      Compute external advantage $\hat{A}_t^{\text{ext}}$ using (5)
    end for
    Asynchronously update policy by minimizing the loss outlined in (10) for each environment
  end while
  Synchronize shared memory to update belief states across environments
  Apply double-buffered sampling to ensure continuous data collection
end while
return trained EPPTA

$$b_t = \frac{\sum_{s_t \in S} b_t(s_t) \cdot s_t}{|S|}, \tag{6}$$

It is evident that b_t symbolizes the mean value of the state distribution under the current belief state. Notably, when the state is entirely known, b_t coincides with the current state.

To better enable agents to implicitly comprehend belief updates, we define a portion of (3) as the transient belief state $\tilde{b}_{t+1}$.

$$\tilde{b}_{t+1} = \sum_{s_t \in S} P(s_{t+1} \mid s_t, a_{t+1}) b_t(s_t), \tag{7}$$

$\tilde{b}_{t+1}$ calculates the expected probability of reaching state s_{t+1} based on the current belief state b_t and the action a_{t+1}. Essentially, it computes the belief state for state s_{t+1} before incorporating the new observation o_{t+1} and represents a summation over all possible states s_t at time t.

Substitute (7) into (3), we have:

$$b_{t+1} = \frac{O(\omega_{t+1} \mid s_{t+1}, a_{t+1}) \tilde{b}_{t+1}}{P(\omega_{t+1} \mid a_{t+1}, b_t)}, \tag{8}$$

Referring to Equation (1), the denominator $P(\omega \mid h, a)$ serves as a normalization constant. We observe that the iterative process of belief updates can be described as $b_t \to \tilde{b}_{t+1} \to b_{t+1}$, where both $b_t \to \tilde{b}_{t+1}$ and $\tilde{b}_{t+1} \to b_{t+1}$ are linear transformations suitable for MLP approximation. Similar to HMM filtering, $b_t \to \tilde{b}_{t+1}$ is akin to the discrete Chapman-Kolmogorov equation in the prediction step, while the computation of $\tilde{b}_{t+1} \to b_{t+1}$ is based on Bayesian estimation, resembling the update step. It is evident that the action execution primarily affects the prediction step, while the update step is predominantly influenced by observations. To maintain consistency with the interaction between the agent and the environment in the RL process, we initiate the iterative process of belief updates commencing from $\tilde{b}_t$ and set the initial transient belief $\tilde{b}_0$ equal to the initial observation ω_0.

The bottom row of Figure 1 illustrates the implicit updates of the belief state, while Figure 2 showcases specific belief state updates within the belief module.

As depicted in Figure 2, the left box represents the belief update process, while the right box represents the belief prediction process. The input variables of the belief module consist of observations and belief states, with the corresponding self-weights denoted as w_ω, w_b, and features f_ω, f_b extracted by adaptive learners with shared parameters. ω_{t+1} and $\tilde{b}_t$ are jointly employed to revise the estimated black-box belief state b_{t+1}. Subsequently, based on the encoded b_{t+1} through f_{b_s} and the encoded action a_{t+1} via f_a, the prediction of the transient belief state $\tilde{b}_{t+1}$ is generated. f_{b_s} and f_a represent distinct neural networks utilized for encoding the features of b_{t+1} and a_{t+1}, respectively. Self-weighted feature extraction of ω_t and b_t informs actions based on these features, yielding a more accurate probability distribution approximation for the current state.

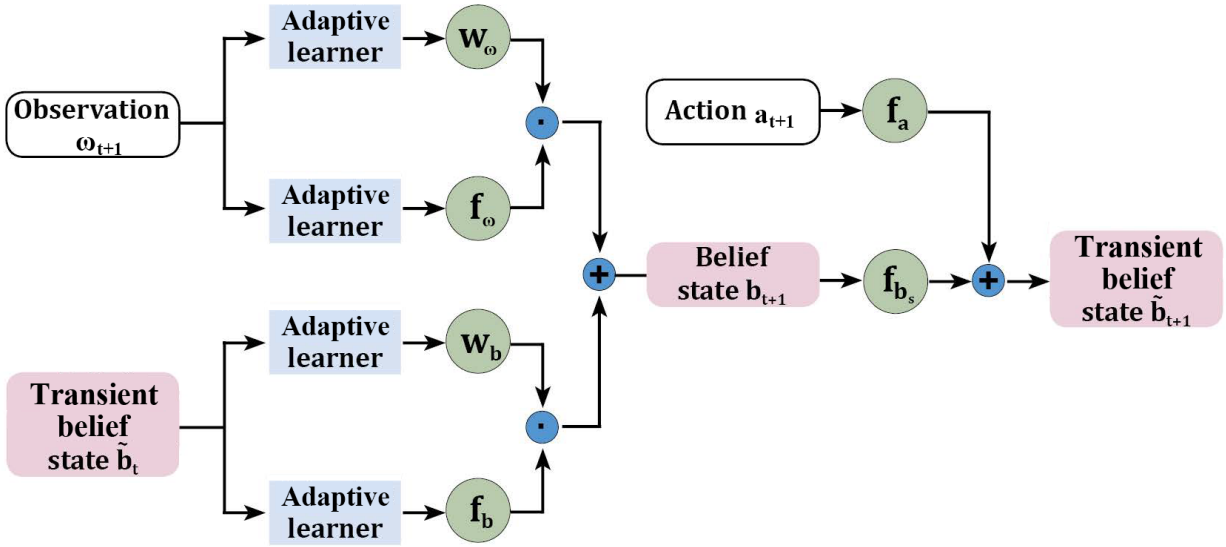

Figure 2 Schematic of the proposed belief module.

The belief update loss function is indispensable as it furnishes a mechanism for the model to learn an approximation of the belief state, even when the true belief state remains unknown. Informed by the interplay between observations, actions, and rewards, the belief update loss function operates as an indirect supervision signal, empowering the model to learn a valuable representation of the environment for decision-making purposes.

Taking inspiration from human beliefs in partially observable situations, we anticipate the agent's beliefs regarding observed aspects to align with the observations, and beliefs about unobserved information to remain consistent with the previous state. Consequently, we define the belief loss function as follows:

$$L^{\text{Belief}} = \sum_t \left(\sum_i \frac{(b_t[i] - o_t[i])^2 * I(o_t[i] \neq 0)}{\sum_i I(o_t[i] \neq 0)} + \sum_i \frac{(b_t[i] - b_{t-1}[i])^2 * I(o_t[i] = 0)}{\sum_i I(o_t[i] = 0)} \right), \tag{9}$$

where I represents the indicator function. In order to simplify the calculation and better fit, we use empirical parameters instead of indicative functions.

The overall loss can be expressed as follows, where α is the weighting parameter:

$$Loss = L^{\text{PPO}} + \alpha * L^{\text{Belief}}. \tag{10}$$

4.2 | Efficiency Enhancements for EPPTA

In this section, we investigate the integration of EPPTA into the RL framework "Sample Factory" to address the following challenges: Firstly, Sample Factory did not initially support the EPPTA algorithm, necessitating innovative modifications and adjustments to enable efficient operation of EPPTA within this framework. Secondly, our focus lies in enhancing the convergence speed and computational resource utilization of EPPTA. Our motivation stems from the extensive potential applications of EPPTA in the field of automated pen-testing, where its performance advantages need to be fully realized in large-scale RL training.

Componentization: We decomposed EPPTA into multiple highly independent components, based on the core principles of the Sample Factory framework: Rollout Workers, Policy Workers, and the Learner. Each component is specialized in performing specific tasks, with Rollout Workers responsible for simulating environments and data collection, Policy Workers dedicated to policy improvement, and the Learner focused on learning from policy updates. These components obtain input data through signal propagation, execute computational tasks, and broadcast computation results through signal emission. By conducting these critical computational tasks asynchronously, we achieve maximum performance enhancement in RL tasks.

Efficient Communication Mechanism: To address real-time communication challenges among the components, we adopted a custom communication mechanism within the Sample Factory framework, inspired by Qt's signal and slot paradigm. This communication mechanism not only facilitates inter-thread communication but also supports inter-process communication and is composed of a collection of EventLoops within the application. Each EventLoop operates as an infinite loop, occupying a thread or process, and is responsible for implementing the system's logic through EventLoopObject components. Multiple EventLoopObjects can be supported by each EventLoop, enabling them to emit signals that include signal names and payloads accommodating arbitrary data. All these components engage in asynchronous interactions through the exchange of signals.

Optimized Data Transmission: To reduce data communication overhead, we opted to employ shared memory buffers for data transmission, such as belief states, rather than explicitly serializing observation data and transferring them across processes. Whenever a component needs to send data to another component, it writes the data into a shared memory buffer and sends a signal containing the buffer ID.

Adaptive Double Buffered Sampling: Uninterrupted acquisition of experiential data holds paramount significance within the realm of RL tasks. To more effectively meet this requirement, we introduced an adaptive double buffered sampling strategy. Given a total of N parallel environments, the original strategy employed by Sample Factory allocated the interaction data of the initial k environments with the agent to the first buffer, with the data from the remaining $N-k$ environments stored in the second buffer. Here, the value of k was determined based on the number of agents completing interactions within a fixed time interval. However, this fixed time interval strategy could result in significant temporal discrepancies in data collection between the two buffers, especially when confronting large-scale tasks characterized by notable variations in interaction times among diverse environments. Importantly, one characteristic of pen-testing tasks is the vast action space, frequently reaching magnitudes of 10^3 or more, leading to substantial discrepancies in interaction times between different agents and environments. In response to this challenge, we introduced an adaptive strategy to dynamically adjust buffer timing. Initially, we estimated the shortest (t_{min}) and longest (t_{max}) interaction times required for different agents and environments across multiple parallel settings. Subsequently, we adapted the data collection time of the first buffer to $(t_{min} + t_{max})/2$. This enhanced approach enables us to minimize the temporal disparities in data collection times between the two buffers, effectively mitigating the potential for load imbalances in double buffered sampling.

The ultimate outcome of these measures is a significant improvement in performance. By closely integrating EPPTA with Sample Factory, we successfully accelerated the convergence rate of training, enhanced training efficiency, and made more effective use of computational resources. These improvements not only showcased EPPTA's exceptional performance in the field of automated pen-testing but also provided a model for research in other areas of RL.

5 | PERFORMANCE EVALUATION

In this section, we present a comprehensive evaluation of the EPPTA framework's performance in various pen-testing scenarios. Our assessment includes comparisons with state-of-the-art RL methods for pen-testing in terms of their efficacy and scalability. We conducted experiments using NASim[9], an open-source platform that provides diverse abstract network scenarios for evaluating pen-testing agents.

5.1 | Experimental Setup

Environments: Our training setup utilized PyTorch version 1.11.0 and an NVIDIA A100 GPU card with 40G memory. We evaluated EPPTA on various network configurations, ranging from small-scale to large-scale scenarios. These configurations include *Tiny*, *Small*, *Medium*, *LargeGen*, *HugeGen*, and *Pocp2Gen*. The smallest environment, *Tiny*, comprises 4 subnets, 3 hosts, an 18-dimensional action space, and a 576-dimensional state space. In contrast, the largest environment, *Pocp2Gen*, features 21 subnets, 95 hosts, a 3515-dimensional action space, and a massive 1.49E+08-dimensional state space. For each of these environments, we executed 12 parallel instances with different random seeds.

Actions available to the agent in NASim include scanning (collecting information on hosts/subnets), exploiting vulnerable services, and privilege escalation (using processes to elevate access). Each action carries the possibility of success or failure, adding an element of uncertainty to the task.

Methods \ Scenarios		Tiny	Small	Medium	LargeGen	HugeGen	Pocp2Gen
HA-DQN	R	184.0 ± 2.7	-256.2 ± 291.1	-2140.4 ± 29.6	-4898.7 ± 85.6	-9695.0 ± 320.2	-29873.2 ± 49.1
	S	16.0 ± 2.7	464.9 ± 87.2	2000 ± 0	5000 ± 0	10000 ± 0	30000 ± 0
	D	8.9(9.9×)	458.9(1147.3×)	–	–	–	–
NDSPI-DQN	R	185.1 ± 2.8	174.0 ± 4.7	167.0 ± 4.3	-4936.4 ± 81.2	-9494.9 ± 427.0	-29764.0 ± 57.2
	S	14.9 ± 2.7	17.3 ± 4.8	20.8 ± 2.9	5000 ± 0	10000 ± 0	30000 ± 0
	D	8.9(9.9×)	11.3(28.3×)	12.8(12.8×)	–	–	–
PPO	R	191.0 ± 0.93	179.4 ± 0.12	-2000 ± 0.71	138.0 ± 50.9	-243.3 ± 67.5	-213.0 ± 87.9
	S	9.1 ± 0.91	12.9 ± 0.15	2000.0 ± 0.0	84.9 ± 45.7	463.7 ± 53.4	413.5 ± 90.3
	D	4.4(4.9×)	4.4(11×)	–	786.0(238.1×)	2577.9(560.5×)	3941.5(1231.7×)
Recurrent PPO	R	**193.2 ± 0.10**	**184.8 ± 0.12**	**182.4 ± 0.23**	208.9 ± 0.34	211.2 ± 0.45	224.6 ± 0.21
	S	**6.7 ± 0.15**	**8.4 ± 0.15**	**9.0 ± 0.16**	14.8 ± 0.33	18.2 ± 0.34	20.1 ± 0.26
	D	**0.7(0.8×)**	**0.4(1×)**	**1.0(1.4×)**	6.8(2.0×)	8.2(1.8×)	8.1(2.3×)
EPPTA (Ours)	R	**193.1 ± 0.26**	**184.9 ± 0.08**	**182.3 ± 2.74**	**211.9 ± 1.23**	**214.3 ± 2.31**	**229.8 ± 3.42**
	S	**6.9 ± 0.07**	**8.4 ± 0.08**	**8.7 ± 2.15**	**11.4 ± 1.47**	**14.5 ± 2.41**	**15.6 ± 4.32**
	D	**0.9(1×)**	**0.4(1×)**	**0.7(1×)**	**3.4(1×)**	**4.5(1×)**	**3.6(1×)**
Oracle	R	194	186	185	217	220	234
	S	6	8	8	8	10	12

Table 1 Learning performance of various methods across different network configurations. IQM episodic rewards and IQM episodic length are denoted as R and S, respectively. The distance between the converged IQM episodic length and the Oracle is represented by D, signifying the additional number of attack operations employed to complete the pen-testing task compared to the theoretical optimal strategy.

The reward function for the pen-testing agent is defined as the value of compromised hosts minus the cost of actions taken. The objective is to compromise all target hosts with positive values on the network while minimizing the number or cost of executed actions.

Evaluation metrics: To benchmark EPPTA's performance, we introduced the "Oracle." The Oracle symbolizes the pinnacle of pen-testing expertise, representing a hypothetical scenario where human experts execute the best actions without errors. Comparing EPPTA's performance to the Oracle provided us with valuable insights into the algorithm's capabilities and its ability to approximate human expertise.

To assess performance, we employ Interquartile Mean (IQM) metrics for episodic rewards and episodic length. IQM provides robustness against outlier scores and exhibits higher statistical efficiency than median and mean metrics. Additionally, we use the distance between the converged IQM episodic length and the Oracle as an auxiliary evaluation metric.

The IQM formula is defined as:

$$IQM(x) = \frac{2\sum_{env \in E} x_{env} * I(Q_1(x_{env}) < x < Q_3(x_{env}))}{|E|}, \tag{11}$$

where x represents the evaluation object, E denotes the set of all parallel environments, I serves as an indicator function, sorting the variables x_{env} in different environments, and Q_1 and Q_3 correspond to the lower and upper quartiles, respectively.

5.2 | Performance Comparison

Table 1 presents a performance comparative analysis of the EPPTA framework in contrast to several contemporary RL methodologies designed for pen-testing, which encompass HA-DQN[16], NDSPI-DQN[17], PPO models[31] and Recurrent PPO models[3]. The table clearly illustrates that EPPTA delivers substantial performance improvements across a spectrum of testing scenarios.

In the context of *Tiny* and *Small* environments, Recurrent PPO models demonstrates the most optimal performance, attributed to its exceptional generalization capabilities, with EPPTA following closely behind. In contrast, RL-based penetration testing

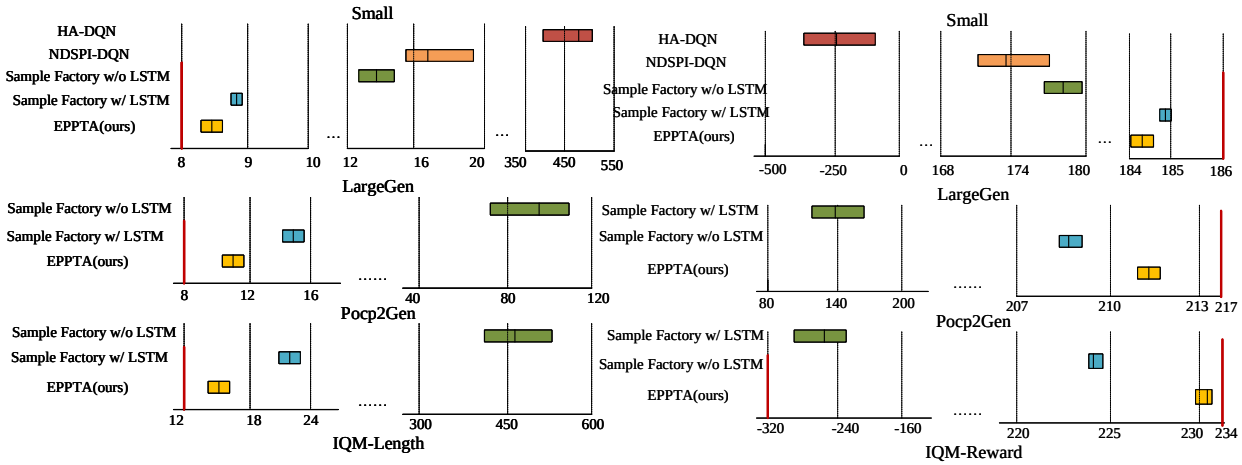

Figure 3 Medium and 95% confidence interval (CI) of IQM episodic reward and IQM episodic length across *Small*, *LargeGen* and *Pocp2Gen* environments.Red vertical lines represent the Oracles.

algorithms such as HA-DQN and NDSPI-DQN exhibit inferior performance, particularly in environments characterized by partial observability.

When scaling up to more extensive scenarios, such as *Medium*, *LargeGen*, *HugeGen*, and *Pocp2Gen*, EPPTA showcases superior performance. It converges more closely to the Oracle's performance and requires significantly fewer actions to attain higher rewards. Although Recurrent PPO models converges to an acceptable performance level in these larger-scale environments, a noticeable gap from the Oracle's performance persists. Moreover, conventional RL-based pen-testing approaches encounter challenges in effectively addressing such extensive scenarios. As discernible from the table, the EPPTA framework outperforms other methods by a margin of at least 1.8×.

Figure 3 depicts a detailed comparison of the methods in three typical NASim scenarios: small-scale *Small*, large-scale *LargeGen*, and huge-scale *Pocp2Gen*. We apply Rliable[33] to present comparisons of the IQM episodic reward and the IQM episodic length between different methods, which confirms the data in Table 1.

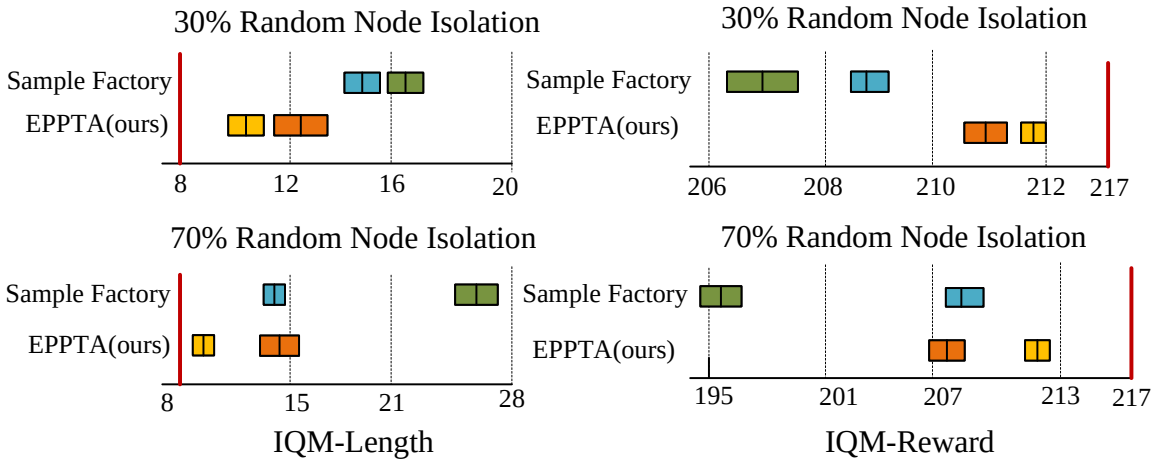

Figure 4 Convergence performance comparison in the presence of random node isolation.

The results reveal that previous RL pen-testing methods primarily concentrate on fully observable networks, performing poorly in partially observable networks and failing to converge in large-scale environments. More general RL methods, such as recurrent PPO models, display relatively stable performance but cannot achieve superior convergence results in large-scale environments. Our proposed EPPTA method integrates a belief module to facilitate the POMDPs. The evaluation based on IQM metrics demonstrates that EPPTA can achieve the most favorable converged results across a diverse range of scenarios.

Furthermore, when a node is deactivated, acquiring information about the node becomes infeasible, even when relevant actions are executed in other subnets and hosts. Consequently, we emulate pen-testing environments in which humans employ random node isolation with diverse probabilities in response to the compromised network. For instance, consider the environment *LargeGen*, which consists of 8 subnets, 23 hosts, a 322-dimensional action space, and a 4.5E6-dimensional state space. Previous RL pen-testing algorithms, such as HA-DQN and NDSPI-DQN, fail to converge in the presence of random human operations.

Figure 4 presents a comparison between recurrent PPO models and EPPTA under node isolation probabilities of 30% and 70%. The yellow and blue boxes represent the performance results from Figure 3 without random node isolation, while the red and green boxes denote the convergence outcomes when random node isolation is implemented. In an environment with 30% random node isolation probability, Recurrent PPO model's reward decreases from 208.9 to 207.3 and its episodic length increase from 14.8 to 16.6, while EPPTA's reward decreases from 211.9 to 210.9 and its episodic length increase from 11.3 to 12.3. When the probability of node isolation is 70%, Recurrent PPO model's reward drops significantly from 208.9 to 197.4 and its episodic length increase from 14.8 to 26.5, whereas EPPTA's reward decreases slightly from 211.9 to 208.5 and its episodic length increase from 11.3 to 14.9. Consequently, the recurrent PPO model-based agent requires, on average, 11.6 extra steps to complete the pen-testing task compared to the EPPTA-based agent, resulting in a 2.7-fold increase in the distance between the episodic length and the Oracle.

Figure 4 demonstrates that EPPTA is more robust in harsh POMDP environments with artificial random node isolation and possesses superior capability to learn implicit belief updates in the pen-testing state space.

5.3 | Convergence Times

In addition to evaluating the convergence times of various pen-testing methods, it is essential to consider the scalability of these methods as network environments grow in complexity and size. Scalability is a crucial aspect of any practical pen-testing tool, as it directly impacts its real-world applicability.

In our investigation, we continued to select four distinct methods for the comparison of convergence time performance, including HA-DQN[16], NDSPI-DQN[17], PPO models[31], and Recurrent PPO models[3]. Notably, for PPO and Recurrent PPO, we leveraged widely adopted high-performance reinforcement learning libraries, Stable Baselines[28], to configure parallelization strategies within multiple environments.

Table 2 provides a comprehensive comparison of the convergence times of the EPPTA framework against other state-of-the-art RL methodologies. It is worth noting that, although EPPTA experiences an increase in the total number of steps to convergence due to its utilization of asynchronous RL frameworks, it consistently surpasses its counterparts in various scenarios, highlighting its remarkable efficiency in achieving convergence.

One of the key takeaways from this analysis is the remarkable scalability of EPPTA. In the *Tiny* and *Small* environments, EPPTA exhibits competitive convergence times, indicating its effectiveness in relatively simpler scenarios. Please note that, although PPO converges the fastest, its convergence results, as evidenced by Table 1, are far from satisfactory. However, EPPTA's standout performance becomes especially evident in the context of larger and intricately structured network configurations, encompassing scenarios like *Medium*, *LargeGen*, *HugeGen*, and *Pocp2Gen*, where EPPTA not only attains convergence but accomplishes this task notably faster compared to alternative methodologies.

In the *Medium* environment, EPPTA's convergence speed is nearly 12.5× faster than PPO and Recurrent PPO. This rapid convergence is essential for timely threat assessment and mitigation in medium-sized networks. In the *LargeGen* and *HugeGen* environments, EPPTA's scalability becomes even more apparent, as it successfully converges while other methods struggle or fail.

Notably, in the challenging *Pocp2Gen* environment, EPPTA's performance is exceptional. It achieves a remarkable 20.04× speedup under partially observable states. This exemplifies EPPTA's capability to efficiently handle large-scale, complex network scenarios that are characteristic of modern cybersecurity challenges.

The scalability of EPPTA can be attributed to its innovative approach in managing real-time communication among its components. Through the utilization of a signal mechanism, the minimization of data transfer overhead via shared memory, and

Methods \ Scenarios		Tiny	Small	Medium	LargeGen	HugeGen	Pocp2Gen
HA-DQN	L	3000	–	–	–	–	–
	T	0.58	–	–	–	–	–
NDSPI-DQN	L	1400	3500	5000	–	–	–
	T	0.39	1.5	1.72	–	–	–
PPO	L	120	120	–	–	–	–
	T	0.0094	0.0089	–	–	–	–
Recurrent PPO	L	120	100	2000	2500	16000	30000
	T	0.027	0.175	1.1	1.5	6.1	49.1
EPPTA	L	**450**	**670**	**2000**	**10000**	**18000**	**35000**
	T	**0.021(1.29×)**	**0.031(5.65×)**	**0.088(12.5×)**	**0.35(4.29×)**	**0.50(12.2×)**	**2.45(20.04×)**

Table 2 Comparison of convergence time among various methods under different network configurations in partially observable states. The columns labeled L represent the converged step count (in thousands), while the columns labeled T represent the convergence time (in hours).The values in parentheses indicate the speedup factor compared to Recurrent PPO, as it is the only method that achieves convergence in all environments.

the optimization of CPU core utilization via adaptive double-buffering, EPPTA can effectively expand its operations to address complex network environments. EPPTA's scalability goes beyond mere convergence times. Its modular architecture permits the seamless integration of supplementary functionalities and the assimilation of emerging pen-testing methodologies. This inherent extensibility guarantees that EPPTA remains adaptable to evolving security challenges and readily incorporates advancements in the field.

EPPTA's superior convergence times and remarkable scalability make it a promising and practical tool for pen-testing in a variety of network environments. Its ability to adapt and perform efficiently across diverse scenarios underscores its potential for real-world cybersecurity applications, where network sizes and complexities continue to evolve.

6 | CONCLUSION

In this work, we have presented EPPTA, an innovative RL framework designed to address the challenges associated with automated penetration testing. Our efforts have been particularly focused on mitigating two primary challenges: the limited observability of scenarios and the considerable training time overhead.

To tackle these challenges, we introduced several novel components within the EPPTA framework. Of particular significance is the development of a belief module, which incorporates transient belief states. This innovation enhances the interaction between the agent's neural network and the environment, resulting in more accurate probability distribution approximations in partial observable environments.

Furthermore, we integrated EPPTA with the Sample Factory framework, a parallelization framework for RL tasks. This integration required adapting EPPTA to the specific requirements of Sample Factory, which traditionally did not support our algorithm. Through this adaptation, we achieved substantial improvements in convergence speed, leveraging the parallelization capabilities of Sample Factory. Key components of the algorithm, including rollout workers, policy workers, and learners, were orchestrated to function asynchronously and independently, optimizing the utilization of available computing resources. Moreover, we addressed the challenge of data collection by adjusting the double-buffered sampling strategy. This approach allowed us to simulate multiple environments concurrently, making better use of CPU cores and GPU resources, ultimately accelerating the training process.

In conclusion, our work demonstrates the adaptability of EPPTA to the Sample Factory framework, enhancing its efficiency and convergence speed. A comprehensive evaluation of EPPTA using NASim demonstrates its superior performance in different scenarios, outperforming current state-of-the-art RL pen-testing methods. Furthermore, EPPTA maintains robust stability in simulated environments that featured human-controlled random node isolation.

By addressing the challenges of automated pen-testing, we have paved the way for more effective and rapid RL-based security assessment methods. The improvements achieved through the integration of EPPTA with Sample Factory underscore the potential for applying similar innovations to other RL domains, thereby advancing the field of RL and its real-world applications. As future work, we envision further exploration of EPPTA's adaptability to various RL applications and the incorporation of more advanced asynchronous communication techniques and implement an intelligent parallel strategy to push the boundaries of RL training efficiency.

AUTHORS AND CONTRIBUTORSHIP

Zegang Li:program optimization;program analysis; original draft writing.Qian Zhang:program optimization; program test.

CONFLICT OF INTEREST

The author declares no potential conflict of interest.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

1. Henry K. *Penetration testing: protecting networks and systems*. IT Governance Publishing . 2012.
2. Schwartz J, Kurniawati H. Autonomous penetration testing using reinforcement learning. *arXiv preprint arXiv:1905.05965* 2019.
3. Ni T, Eysenbach B, Salakhutdinov R. Recurrent model-free rl can be a strong baseline for many pomdps. In: PMLR. ; 2022: 16691–16723.
4. Madan K, Ke NR, Goyal A, Schölkopf B, Bengio Y. Fast and slow learning of recurrent independent mechanisms. *arXiv preprint arXiv:2105.08710* 2021.
5. Gregor K, Jimenez Rezende D, Besse F, Wu Y, Merzic H, Oord v. dA. Shaping belief states with generative environment models for rl. *Advances in Neural Information Processing Systems* 2019; 32.
6. Chen X, Mu YM, Luo P, Li S, Chen J. Flow-based Recurrent Belief State Learning for POMDPs. In: PMLR. ; 2022: 3444–3468.
7. Hochreiter S, Schmidhuber J. Long short-term memory. *Neural Comput.* 1997; 9(8): 1735–1780.
8. Petrenko A, Huang Z, Kumar T, Sukhatme G, Koltun V. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In: PMLR. ; 2020: 7652–7662.
9. Schwartz J. Network Attack Simulator. https://github.com/Jjschwartz/NetworkAttackSimulator; 2020.
10. Applebaum A, Miller D, Strom B, Korban C, Wolf R. Intelligent, automated red team emulation. In: ; 2016: 363–373.
11. Lallie HS, Debattista K, Bal J. A review of attack graph and attack tree visual syntax in cyber security. *Computer Science Review* 2020; 35: 100219.
12. Gangupantulu R, Cody T, Rahma A, Redino C, Clark R, Park P. Crown Jewels Analysis using Reinforcement Learning with Attack Graphs. In: IEEE. ; 2021: 1–6.

13. Zennaro FM, Erdodi L. Modeling penetration testing with reinforcement learning using capture-the-flag challenges: trade-offs between model-free learning and a priori knowledge. *arXiv preprint arXiv:2005.12632* 2020.

14. Ghanem MC, Chen TM. Reinforcement learning for efficient network penetration testing. *Information* 2019; 11(1): 6.

15. Nguyen HV, Teerakanok S, Inomata A, Uehara T. The Proposal of Double Agent Architecture using Actor-critic Algorithm for Penetration Testing.. In: ; 2021: 440–449.

16. Tran K, Akella A, Standen M, et al. Deep hierarchical reinforcement agents for automated penetration testing. *arXiv preprint arXiv:2109.06449* 2021.

17. Zhou S, Liu J, Hou D, Zhong X, Zhang Y. Autonomous Penetration Testing Based on Improved Deep Q-Network. *Applied Sciences* 2021; 11(19): 8823.

18. Hu Z, Beuran R, Tan Y. Automated Penetration Testing Using Deep Reinforcement Learning. *Proceedings - 5th IEEE European Symposium on Security and Privacy Workshops, Euro S and PW 2020* 2020: 2–10. doi: 10.1109/EuroSPW51379.2020.00010

19. Pineau J, Gordon G, Thrun S, others . Point-based value iteration: An anytime algorithm for POMDPs. In: . 3. ; 2003: 1025–1032.

20. Bai H, Hsu D, Lee WS, Ngo VA. Monte Carlo value iteration for continuous-state POMDPs. In: Springer. ; 2011: 175–191.

21. Chung J, Gülçehre Ç, Cho K, Bengio Y. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR* 2014; abs/1412.3555.

22. Hausknecht M, Stone P. Deep recurrent q-learning for partially observable mdps. In: ; 2015.

23. Gangwani T, Lehman J, Liu Q, Peng J. Learning belief representations for imitation learning in pomdps. In: PMLR. ; 2020: 1061–1071.

24. Park G, Choi S, Sung Y. Blockwise Sequential Model Learning for Partially Observable Reinforcement Learning. In: AAAI Press; 2022: 7941–7948.

25. Mnih, V., Badia, A.P., Mirza, M., Graves, et al. Asynchronous methods for deep reinforcement learning. In: PMLR. ; 2016: 1928–1937.

26. Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J. and Kautz, J. GA3C: GPU-based A3C for deep reinforcement learning. *CoRR* 2016; abs/1611.06256.

27. Espeholt L, Soyer H, Munos R, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In: PMLR. ; 2018: 1407–1416.

28. Hill A, Raffin A, Ernestus M, et al. Stable Baselines. https://github.com/hill-a/stable-baselines; 2018.

29. Liang E, Liaw R, Nishihara R, et al. RLlib: Abstractions for distributed reinforcement learning. In: PMLR. ; 2018: 3053–3062.

30. Strom BE, Applebaum A, Miller DP, Nickels KC, Pennington AG, Thomas CB. Mitre att&ck: Design and philosophy. *Mitre Product Mp* 2018: 18–0944.

31. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* 2017.

32. Schulman J, Moritz P, Levine S, Jordan M, Abbeel P. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* 2015.

33. Agarwal R, Schwarzer M, Castro PS, Courville AC, Bellemare M. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems* 2021; 34: 29304–29320.

Appendices

FIGURES AND TABLES

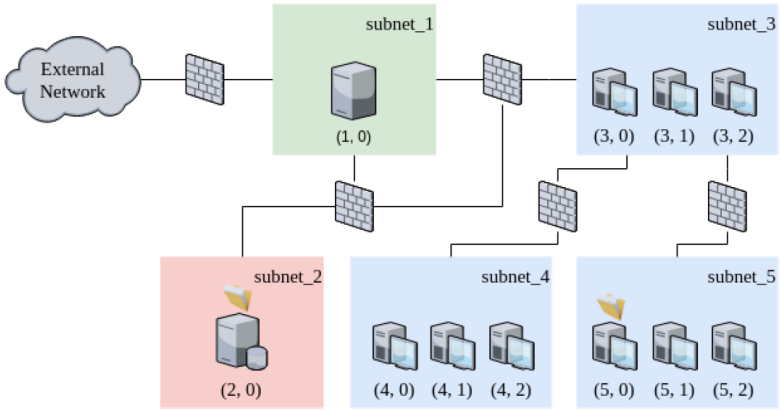

Figure 5 Example network with 5 subnets, 11 machines and the sensitive documents located on machines (2, 0) and (5, 0). (Schwartz,2019)

Name	Type	Subnets	Hosts	OS	Services	Processes	Exploits	PrivEscs	Actions
Tiny	Static	4	3	1	1	1	1	1	18
Small	Static	5	8	2	3	2	3	2	72
Medium	Static	6	16	2	5	3	5	3	192
Large-Gen	Generated	8	23	3	7	3	7	3	322
Huge-Gen	Generated	11	38	4	10	4	10	4	684
Pocp-2-Gen	Generated	21	95	3	10	3	30	3	3515
Observation Dims	States	Step Limit							
4*14	576	1000							
9*23	24576	1000							
17*27	393216	2000							
24*32	4521984	5000							
39*40	2.39E+08	10000							
96*48	1.49E+08	30000							

Table 3 NASim Benchmark Scenario Configurations and Parameters

The action space encompasses various types of actions, such as exploitation, privilege escalation, scanning, and more. Each action is associated with specific costs and success probabilities. For instance, action number 4 represents an exploitation attempt on the 0th host of the 1st subnet, with a cost of 3 and a success probability of 0.9, resulting in obtaining primary access to a Linux system. Consequently, the minimum number of steps achievable by a human expert, referred to as the oracle, does not necessarily represent the optimal number of steps that RL methods can attain. The expected episodic length for RL methods can be estimated as follows, using the Small environment as an example: Let p_a denote the success probability of a performed action a. The expected number of steps required for this action is represented as:

```
============================================================
0 ServiceScan: target=(1, 0), cost=1.00, prob=1.00, req_access=USER
1 OSScan: target=(1, 0), cost=1.00, prob=1.00, req_access=USER
2 SubnetScan: target=(1, 0), cost=1.00, prob=1.00, req_access=USER
3 ProcessScan: target=(1, 0), cost=1.00, prob=1.00, req_access=USER
4 Exploit: target=(1, 0), cost=3.00, prob=0.90, req_access=USER, os=linux, service=ssh, access=1
5 Exploit: target=(1, 0), cost=1.00, prob=0.60, req_access=USER, os=windows, service=ftp, access=1
6 Exploit: target=(1, 0), cost=2.00, prob=0.90, req_access=USER, os=None, service=http, access=1
7 PrivilegeEscalation: target=(1, 0), cost=1.00, prob=1.00, req_access=USER, os=linux, process=tomcat, access=2
8 PrivilegeEscalation: target=(1, 0), cost=1.00, prob=1.00, req_access=USER, os=windows, process=daclsvc, access=2
9 ServiceScan: target=(2, 0), cost=1.00, prob=1.00, req_access=USER
10 OSScan: target=(2, 0), cost=1.00, prob=1.00, req_access=USER
11 SubnetScan: target=(2, 0), cost=1.00, prob=1.00, req_access=USER
12 ProcessScan: target=(2, 0), cost=1.00, prob=1.00, req_access=USER
13 Exploit: target=(2, 0), cost=3.00, prob=0.90, req_access=USER, os=linux, service=ssh, access=1
14 Exploit: target=(2, 0), cost=1.00, prob=0.60, req_access=USER, os=windows, service=ftp, access=1
15 Exploit: target=(2, 0), cost=2.00, prob=0.90, req_access=USER, os=None, service=http, access=1
16 PrivilegeEscalation: target=(2, 0), cost=1.00, prob=1.00, req_access=USER, os=linux, process=tomcat, access=2
17 PrivilegeEscalation: target=(2, 0), cost=1.00, prob=1.00, req_access=USER, os=windows, process=daclsvc, access=2
18 ServiceScan: target=(3, 0), cost=1.00, prob=1.00, req_access=USER
19 OSScan: target=(3, 0), cost=1.00, prob=1.00, req_access=USER
20 SubnetScan: target=(3, 0), cost=1.00, prob=1.00, req_access=USER
21 ProcessScan: target=(3, 0), cost=1.00, prob=1.00, req_access=USER
22 Exploit: target=(3, 0), cost=3.00, prob=0.90, req_access=USER, os=linux, service=ssh, access=1
23 Exploit: target=(3, 0), cost=1.00, prob=0.60, req_access=USER, os=windows, service=ftp, access=1
24 Exploit: target=(3, 0), cost=2.00, prob=0.90, req_access=USER, os=None, service=http, access=1
25 PrivilegeEscalation: target=(3, 0), cost=1.00, prob=1.00, req_access=USER, os=linux, process=tomcat, access=2
26 PrivilegeEscalation: target=(3, 0), cost=1.00, prob=1.00, req_access=USER, os=windows, process=daclsvc, access=2
27 ServiceScan: target=(3, 1), cost=1.00, prob=1.00, req_access=USER
28 OSScan: target=(3, 1), cost=1.00, prob=1.00, req_access=USER
29 SubnetScan: target=(3, 1), cost=1.00, prob=1.00, req_access=USER
30 ProcessScan: target=(3, 1), cost=1.00, prob=1.00, req_access=USER
31 Exploit: target=(3, 1), cost=3.00, prob=0.90, req_access=USER, os=linux, service=ssh, access=1
32 Exploit: target=(3, 1), cost=1.00, prob=0.60, req_access=USER, os=windows, service=ftp, access=1
33 Exploit: target=(3, 1), cost=2.00, prob=0.90, req_access=USER, os=None, service=http, access=1
34 PrivilegeEscalation: target=(3, 1), cost=1.00, prob=1.00, req_access=USER, os=linux, process=tomcat, access=2
35 PrivilegeEscalation: target=(3, 1), cost=1.00, prob=1.00, req_access=USER, os=windows, process=daclsvc, access=2
36 ServiceScan: target=(3, 2), cost=1.00, prob=1.00, req_access=USER
37 OSScan: target=(3, 2), cost=1.00, prob=1.00, req_access=USER
38 SubnetScan: target=(3, 2), cost=1.00, prob=1.00, req_access=USER
39 ProcessScan: target=(3, 2), cost=1.00, prob=1.00, req_access=USER
40 Exploit: target=(3, 2), cost=3.00, prob=0.90, req_access=USER, os=linux, service=ssh, access=1
41 Exploit: target=(3, 2), cost=1.00, prob=0.60, req_access=USER, os=windows, service=ftp, access=1
42 Exploit: target=(3, 2), cost=2.00, prob=0.90, req_access=USER, os=None, service=http, access=1
43 PrivilegeEscalation: target=(3, 2), cost=1.00, prob=1.00, req_access=USER, os=linux, process=tomcat, access=2
44 PrivilegeEscalation: target=(3, 2), cost=1.00, prob=1.00, req_access=USER, os=windows, process=daclsvc, access=2
45 ServiceScan: target=(3, 3), cost=1.00, prob=1.00, req_access=USER
46 OSScan: target=(3, 3), cost=1.00, prob=1.00, req_access=USER
47 SubnetScan: target=(3, 3), cost=1.00, prob=1.00, req_access=USER
48 ProcessScan: target=(3, 3), cost=1.00, prob=1.00, req_access=USER
49 Exploit: target=(3, 3), cost=3.00, prob=0.90, req_access=USER, os=linux, service=ssh, access=1
50 Exploit: target=(3, 3), cost=1.00, prob=0.60, req_access=USER, os=windows, service=ftp, access=1
51 Exploit: target=(3, 3), cost=2.00, prob=0.90, req_access=USER, os=None, service=http, access=1
52 PrivilegeEscalation: target=(3, 3), cost=1.00, prob=1.00, req_access=USER, os=linux, process=tomcat, access=2
53 PrivilegeEscalation: target=(3, 3), cost=1.00, prob=1.00, req_access=USER, os=windows, process=daclsvc, access=2
54 ServiceScan: target=(3, 4), cost=1.00, prob=1.00, req_access=USER
55 OSScan: target=(3, 4), cost=1.00, prob=1.00, req_access=USER
56 SubnetScan: target=(3, 4), cost=1.00, prob=1.00, req_access=USER
57 ProcessScan: target=(3, 4), cost=1.00, prob=1.00, req_access=USER
58 Exploit: target=(3, 4), cost=3.00, prob=0.90, req_access=USER, os=linux, service=ssh, access=1
59 Exploit: target=(3, 4), cost=1.00, prob=0.60, req_access=USER, os=windows, service=ftp, access=1
60 Exploit: target=(3, 4), cost=2.00, prob=0.90, req_access=USER, os=None, service=http, access=1
61 PrivilegeEscalation: target=(3, 4), cost=1.00, prob=1.00, req_access=USER, os=linux, process=tomcat, access=2
62 PrivilegeEscalation: target=(3, 4), cost=1.00, prob=1.00, req_access=USER, os=windows, process=daclsvc, access=2
63 ServiceScan: target=(4, 0), cost=1.00, prob=1.00, req_access=USER
64 OSScan: target=(4, 0), cost=1.00, prob=1.00, req_access=USER
65 SubnetScan: target=(4, 0), cost=1.00, prob=1.00, req_access=USER
66 ProcessScan: target=(4, 0), cost=1.00, prob=1.00, req_access=USER
67 Exploit: target=(4, 0), cost=3.00, prob=0.90, req_access=USER, os=linux, service=ssh, access=1
68 Exploit: target=(4, 0), cost=1.00, prob=0.60, req_access=USER, os=windows, service=ftp, access=1
69 Exploit: target=(4, 0), cost=2.00, prob=0.90, req_access=USER, os=None, service=http, access=1
70 PrivilegeEscalation: target=(4, 0), cost=1.00, prob=1.00, req_access=USER, os=linux, process=tomcat, access=2
71 PrivilegeEscalation: target=(4, 0), cost=1.00, prob=1.00, req_access=USER, os=windows, process=daclsvc, access=2
------------------------------------------------------------
```

Figure 6 The action space of *Small*

$$E_a = \sum_{i=1}^{\infty} i * (1 - p_a)^{i-1} * p_a = \frac{1}{p_a}$$

Considering an optimal policy for the Small environment as an example, which follows the action trajectory [6, 2, 13, 16, 33, 29, 67, 70], the oracle's value is 8. However, the expectation for RL methods should be:

$$E_{RL} = \sum_{i=1}^{8} E_{a_i}$$

where E_{a_i} represents the expected number of steps for each action in the trajectory, and this value may exceed the oracle's count.

Scenario	Mlp-layers-size	RNN-size	Learning rate	IQM reward	Reward StdDev	IQM length	Length StdDev
Tiny	**32**	**128**	**0.0012**	**193.07**	**0.26**	**6.88**	**0.07**
	32	128	0.0015	192.15	0.45	7.29	0.09
	64	128	0.0012	189.63	0.79	16.93	0.43
	128	256	0.0012	182.09	1.05	19.22	0.74
Small	**64**	**128**	**0.0015**	**184.94**	**0.08**	**8.40**	**0.08**
	32	128	0.0015	183.80	0.08	8.84	0.24
	64	128	0.0012	183.60	0.83	9.21	0.39
Medium	**32**	**128**	**0.0012**	**182.32**	**2.74**	**8.74**	**2.15**
	64	128	0.0015	179.09	2.34	10.98	2.20
LargeGen	**32**	**128**	**0.0012**	**211.88**	**1.23**	**11.41**	**1.47**
	32	128	0.0015	205.07	7.42	19.21	5.13
	64	128	0.0012	210.16	3.36	11.99	3.48
	64	128	0.0015	209.67	4.34	12.37	4.51
HugeGen	**16**	**128**	**0.0012**	**214.28**	**2.31**	**14.51**	**2.41**
	32	64	0.0012	213.04	4.90	15.31	4.06
Pocp2Gen	**16**	**128**	**0.0012**	**229.83**	**3.42**	**15.61**	**4.32**
	32	256	0.0015	227.18	3.91	17.09	2.63

Table 4 Comparison of convergence performance between different algorithms on various scenarios in terms of Mlp-layers-size, RNN-size, Learning rate, IQM reward, and IQM length.

Methods \ Scenarios		Tiny	Small	Medium	LargeGen	HugeGen	Pocp2Gen
HA-DQN	L	1000	3000	–	–	–	–
	T	0.49	2.15	–	–	–	–
NDSPI-DQN	L	800	1600	1800	–	–	–
	T	0.35	1.3	1.37	–	–	–
PPO	L	65	100	3000	2000	4500	6000
	T	0.0083	0.011	0.0250	0.136	0.317	0.417
Recurrent PPO	L	66	100	3000	3000	15000	25000
	T	0.02	0.028	0.167	0.72	4.3	40.1
EPPTA	L	**350**	**615**	**800**	**2000**	**16000**	**30000**
	T	**0.0108(1.85×)**	**0.0183(1.53×)**	**0.0242(6.90×)**	**0.0511(14.09×)**	**0.41(10.49×)**	**2.2(18.23×)**

Table 5 Comparison of convergence time among various methods under different network configurations in fully observable states.The columns labeled L represent the converged step count (in thousands), while the columns labeled T represent the convergence time (in hours).

How to cite this article: Zegang Li, Qian Zhang, Guangwen Yang (2023), EPPTA: Efficient Partially Observable Reinforcement Learning Agent for Penetration testing Applications, *Engineering Reports*, .