

Article

L-ARLPT: An LLM-Augmented Reinforcement Learning Framework for Autonomous Penetration Testing

Rufeng Zhan ¹, Junyi Zhu ^{2,3}, Yinghui Xu ¹, Chan Chen ¹, Rongxin Hu ¹ and Le Wang ^{1,*}

¹ Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou 510700, China; 2006400027@e.gzhu.edu.cn (R.Z.); xuyinghui@e.gzhu.edu.cn (Y.X.); chenchan01@e.gzhu.edu.cn (C.C.); hurongxin@e.gzhu.edu.cn (R.H.)

² Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China; 12547014@mail.sustech.edu.cn

³ Department of New Networks, Peng Cheng Laboratory, Xili Street, Shenzhen 518000, China

* Correspondence: wangle@gzhu.edu.cn

Abstract

In recent years, Deep Reinforcement Learning (DRL) has emerged as a promising approach for automating penetration testing due to its capability to perform sequential decision-making in complex environments. However, in real-world enterprise networks, attack actions are typically characterized by highly coupled multi-dimensional parameter combinations, resulting in an exponentially expanding discrete action space. Such a large action space significantly degrades exploration efficiency and prevents conventional DRL agents from learning effective attack paths under sparse-reward conditions. To address these challenges, this paper proposes a Large Language Model-enhanced Autonomous Reinforcement Learning Penetration Testing framework (L-ARLPT). Specifically, the framework leverages the domain knowledge embedded in a Large Language Model (LLM) to perform tactical planning, thereby pruning the original action space into a compact set of candidate actions. Subsequently, an experience-driven layer employs the optimization mechanism of a Deep Q-Network (DQN) to conduct value estimation and policy learning within the reduced candidate set. To validate the effectiveness of the proposed framework, a high-fidelity enterprise penetration-testing simulation environment was constructed based on realistic enterprise attack scenarios. Experimental results demonstrate that, in a high-fidelity enterprise penetration-testing environment with a raw theoretical parameter-combination space containing 6×10^7 combinations, the proposed L-ARLPT framework achieves an average penetration depth of 3.33 out of 4.00, substantially outperforming both reinforcement learning baselines (all ≤ 1.27) and LLM-based baselines (1.12). Moreover, successful episodes require only 128.43 decision steps on average, enabling long-horizon, cross-domain lateral penetration in high-dimensional discrete action spaces.

Keywords: automated penetration testing; reinforcement learning; large language model; prompt engineering; action space pruning



Academic Editor: George Drosatos

Received: 11 June 2026

Revised: 5 August 2026

Accepted: 6 August 2026

Published: 7 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons](#)

[Attribution \(CC BY\)](#) license.

1. Introduction

As cybersecurity threats continue to evolve, red teaming has become an important approach for evaluating the effectiveness of defensive systems [1]. However, its strong dependence on human expertise incurs substantial labor and time costs, limiting scalability in practice [2,3]. To address this limitation, automated penetration testing has attracted increasing attention. Among existing approaches, DRL is particularly promising

because it can learn complex attack strategies through continuous interaction with the environment [4–6]. The action space in real-world penetration testing environments often grows exponentially, and handling large action spaces has been identified as one of the core challenges faced by autonomous penetration testing agents [7,8]. Moreover, strict prerequisite dependencies among attack stages lead to sparse effective feedback, further increasing the difficulty of reinforcement learning-based attack planning [9].

To address the exploration inefficiency caused by high-dimensional combinatorial action spaces, existing studies can be broadly categorized into four research directions. The first category addresses large discrete action spaces through continuous action representations. Representative studies employ the Wolpertinger architecture [10,11], which maps discrete attack actions into a continuous embedding space and retrieves candidate actions via nearest-neighbor search. More recently, APRIL further improves scalability by learning transferable semantic action embeddings, enabling efficient policy transfer across different penetration-testing environments while alleviating the optimization difficulty caused by large discrete action spaces [12]. Although such methods improve scalability in large action spaces, the additional retrieval process may reduce policy optimization efficiency. The second category focuses on action decomposition. Representative approaches include the hierarchical CRLA framework [13] and dimension-wise decomposition methods such as Action Branching [14] and ABQ [15], and more recent hierarchical penetration-testing frameworks based on action clipping and decomposition [16], which further reduce exploration complexity by combining action decomposition with reward-guided action pruning. By decomposing combinatorial action spaces into smaller decision units, these methods reduce search complexity. However, they often face challenges in modeling inter-action dependencies and maintaining semantic consistency among decomposed action components.

The third category attempts to filter infeasible actions using prior knowledge or auxiliary models. Representative studies employ MulVAL-based attack dependency graphs [17–19] to pre-screen candidate actions. While these approaches reduce ineffective exploration to some extent, they generally rely on static rules or simple models and therefore exhibit limited adaptability to dynamic environments and complex attack scenarios. The fourth category improves the exploration mechanism of reinforcement learning agents. For instance, DUSC-DQN [20] combines historical information with upper-confidence-bound exploration to improve sample efficiency. Nevertheless, such methods do not fundamentally reduce the size of the action space and therefore remain computationally expensive in large combinatorial environments. In summary, existing studies have improved exploration efficiency through action representation learning, action decomposition, rule-based constraints, and exploration-strategy optimization. However, most approaches still rely heavily on trial-and-error [21,22] learning and cannot effectively exploit cybersecurity domain knowledge to fundamentally reduce the search space. More recently, researchers have begun exploiting the prior knowledge embedded in LLMs to guide reinforcement learning for automated penetration testing. Representative approaches employ LLM-generated action constraints or masking rules to reduce ineffective exploration and improve training efficiency [21]. While existing methods typically use offline, fixed LLM-derived constraints, our framework performs step-wise online semantic reasoning. At each decision step, the LLM analyzes the current context (including new observations and historical feedback) and dynamically reconstructs an executable candidate action set. Rather than relying on precomputed rules, we continuously incorporate environmental feedback into subsequent reasoning cycles via contextual memory. Consequently, efficiently integrating domain knowledge with reinforcement learning remains an open challenge [23] in realistic

penetration-testing environments characterized by high-dimensional combinatorial action spaces, strict prerequisite dependencies, and sparse rewards.

Recent advances in large language models (LLMs) have opened new opportunities for automated penetration testing by leveraging extensive domain knowledge and powerful reasoning capabilities [24]. Nevertheless, their reliance on static pretraining corpora makes them vulnerable to factual hallucinations, while the absence of an environment-interaction-driven value optimization mechanism limits their ability to adapt to dynamically evolving environments and maximize long-term attack effectiveness [25,26]. Motivated by these observations, this work integrates the reasoning capability of LLMs with the long-term value optimization ability of DRL and proposes L-ARLPT, a framework for autonomous penetration testing in high-dimensional action spaces. The main contributions of this work are summarized as follows:

- We propose L-ARLPT, an LLM-augmented reinforcement learning framework designed for autonomous penetration testing in large combinatorial action spaces. The framework employs a knowledge-driven layer to perform two-stage semantic action-space pruning based on cybersecurity knowledge and attack-dependency constraints, dynamically reducing the original action space to a small set of high-quality candidate actions. An experience-driven layer then performs value estimation and policy learning within the candidate set using DQN, reducing ineffective exploration while improving long-horizon attack-path discovery.
- We design a dynamic memory management mechanism with separated positive and negative feedback. To address the tendency of LLMs to become trapped in repetitive trial-and-error loops during multi-step decision-making, the proposed mechanism independently manages sliding-window negative feedback and global positive feedback, enabling online knowledge correction and hallucination suppression while reducing computational resources wasted on ineffective attempts.
- We construct a controlled enterprise penetration-testing simulation platform comprising ten representative business nodes distributed across four logical security zones. Following realistic enterprise network architectures, red-team tactics such as reconnaissance, credential theft, privilege escalation, and cross-domain lateral movement are formalized as POMDP tasks with strict causal dependencies. Although compact in node count, the platform preserves the sparse-reward characteristics, multi-stage prerequisite dependencies, and long-horizon decision-making challenges targeted in this study, thereby providing a standardized and reproducible benchmark for evaluating intelligent attack agents in high-dimensional action spaces.

The remainder of this paper is organized as follows. Section 2 formulates the penetration testing problem as a POMDP and describes the simulation environment. Section 3 presents the proposed L-ARLPT framework. Section 4 reports the experimental results and ablation studies. Section 5 concludes the paper and discusses future work.

2. Simulation Environment Design and Problem Formulation

This study constructs a penetration-testing simulation environment based on a realistic enterprise network and formulates it as a Partially Observable Markov Decision Process (POMDP) [27] with multiple isolated security domains.

2.1. Problem Formulation

In automated penetration testing, attackers generally have no prior knowledge of the complete network configuration and must perform sequential decision-making based solely on limited local observations. To characterize this process, the automated penetration

testing task is formalized as a POMDP defined by the six-tuple (S, A, T, R, O, Ω) , where each component is specified as follows.

1. **State Space (S):** The global state $s \in S$ encodes both the system attributes and asset attributes of all network nodes. System attributes include operating systems, open ports, running services, and existing vulnerabilities, whereas asset attributes include user privileges, credentials, sensitive files, and proxy tunnels.
2. **Action Space (A):** Each attack action is represented as a parameterized combination of multiple components, such as attack type, target entity, credentials, and tools. The Cartesian-product formulation results in a large discrete action space with complex dependencies among action components.
3. **State Transition Function (T):** State transitions in the proposed environment are deterministic. If an action a satisfies the prerequisite conditions under state s , such as network connectivity, access privileges, and credential availability, the environment transitions to a new state with probability 1. Otherwise, the action fails and the state remains unchanged.
4. **Reward Function (R):** The reward function provides an immediate scalar feedback signal to the agent when a state transition occurs. Since only a small fraction of actions are effective within the high-dimensional action space, the resulting reward signal is highly sparse.
5. **Observation Space (Ω) and Observation Function (O):** The agent can only access partial observations. An observation $o_t \in \Omega$ consists of information obtained through previous attack actions, including discovered system attributes and asset attributes. Therefore, the observation function can be formulated as:

$$o_t = O(s_t, H_t) \quad (1)$$

where s_t denotes the complete environment state and H_t represents the historical interaction trajectory available to the agent. The observation function extracts only the currently discovered information from the hidden global state rather than exposing the complete state representation. As attack actions reveal additional resources, vulnerabilities, and credentials, the observation evolves over time, enabling the agent to gradually increase its knowledge about the environment.

In the current implementation, state transitions are deterministic to support controlled and reproducible evaluation of predefined penetration-testing scenarios. Real enterprise networks may exhibit stochasticity due to network latency, service instability, defensive mechanisms, and concurrent activities; extending the framework to stochastic transition models is therefore an important direction for future work. Overall, the proposed POMDP captures the high-dimensional combinatorial action space, partial observability, and sparse feedback characteristics of automated penetration testing.

2.2. Simulation Environment Design

Real-world penetration testing environments are characterized by uncontrollable state transitions and the absence of standardized interaction interfaces, making them unsuitable for the iterative training and evaluation of reinforcement learning algorithms. To address these limitations, a red-team internal network penetration simulation platform was developed. The platform is designed to provide a controlled and reproducible evaluation environment that instantiates the POMDP model defined in Section 2.1. By reproducing realistic attack dependencies and long-horizon decision processes encountered in automated penetration testing, the platform provides a practical basis for evaluating autonomous decision-making approaches.

2.2.1. Network Topology and State Space Instantiation

To construct global state sequences with deep causal dependencies, the simulation platform adopts a realistic enterprise architecture consisting of an external Internet segment and four internal logical security domains. The resulting topology contains ten critical business nodes, as illustrated in Figure 1. In the proposed POMDP formulation, the global state is instantiated as the collection of multi-dimensional attributes associated with all network nodes. Specifically, the states of each node is characterized by two categories of information:

- System attributes: IP address, operating system, open ports, running services, and known vulnerabilities.
- Acquired assets: user credentials, uploaded attack tools, and discovered sensitive files. User credentials and file assets are further categorized according to their associated entities (physical hosts or service layers) to explicitly model the prerequisite dependencies required for privilege escalation and lateral movement.

The benchmark contains ten representative business nodes distributed across four logical security zones: three nodes in the DMZ, three in the Office zone, two in the Operations zone, and two in the Core Production zone. Its complexity is therefore determined not by node count alone, but by the joint effects of network segmentation, state representation, combinatorial actions, and causal dependencies. Specifically, the observation representation covers discovered hosts, vulnerabilities, credentials, files, deployed tools, and attack-failure history. The action representation contains eight parameter fields, whose unconstrained Cartesian product yields 60,374,160 raw theoretical combinations. Only a small fraction of these combinations satisfy action-template compatibility and observation-dependent prerequisites, including network reachability, credential availability, privilege, and tool deployment. Moreover, reaching the final asset in the Core Production zone requires the agent to progress from the externally exposed DMZ through multiple trust boundaries and attack stages. Therefore, although compact in node count, the environment is sufficient for the controlled evaluation of the cross-domain causal reasoning and long-horizon action-selection challenges targeted in this work.

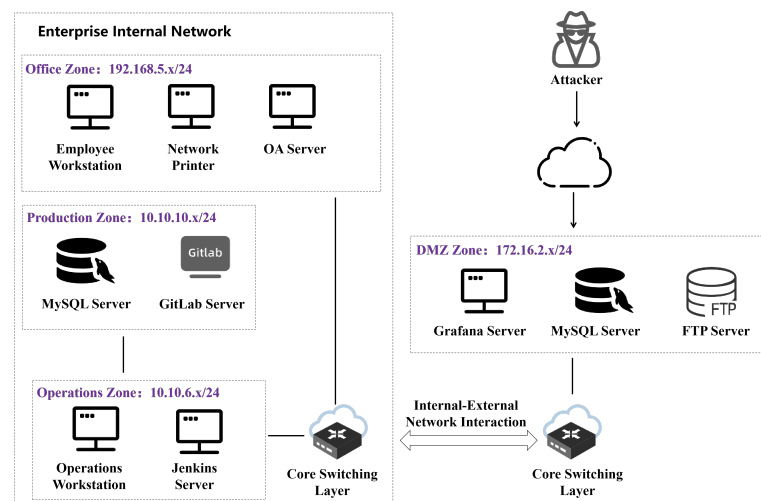


Figure 1. Network Topology.

2.2.2. Action Space Instantiation

In the proposed environment, each attack action is defined as a parameterized combination $a = \{p_1, p_2, \dots, p_n\}$, where the parameter values originate from two distinct sources. The first source consists of environment-specific information dynamically acquired by

the agent during interaction, such as discovered IP addresses, compromised credentials, and identified exploitable vulnerabilities. The second source consists of prior knowledge stored in the agent's static knowledge base, including commonly used attack tools and frequently observed weak-password patterns. The mapping from the 22 actionName values to their required parameter fields, operational semantics, and action categories is provided in Appendix B.

In penetration testing scenarios, action validity is constrained by two factors. The first is semantic consistency among action parameters, requiring all parameter combinations within an action to satisfy logical compatibility constraints. For example, when using a directory-scanning tool (e.g., dirsearch), the action type must correspond to keyword_scan; when invoking an offline password-cracking tool (e.g., john_the_ripper), a target file (e.g., an id_rsa private key) must be specified; and when performing vulnerability exploitation, the selected tool must be compatible with the target vulnerability. The second constraint concerns environmental prerequisites, whereby the current observation must satisfy the conditions required for action execution. Examples include the existence of a reachable vulnerable target, the availability of the required attack tools on the execution host, and possession of valid user credentials. Any action violating either constraint fails to trigger a valid state transition and results only in a negative reward.

The global discrete action space used by the reinforcement-learning agents is defined as the Cartesian product of eight selectable parameter domains: actionName, networkIp, tool, word, userName, passwd, softwareName, and phyFileName. Their cardinalities in the current benchmark are 22, 11, 7, 5, 11, 9, 8, and 9, respectively. Although the benchmark contains ten hosts, the networkIp domain contains eleven distinct values because one dual-homed host has both an external and an internal IP address. Therefore, the global action-space cardinality is:

$$\begin{aligned}
 |\mathcal{A}| &= |P_{\text{actionName}}| |P_{\text{networkIp}}| |P_{\text{tool}}| |P_{\text{word}}| |P_{\text{userName}}| |P_{\text{passwd}}| |P_{\text{softwareName}}| |P_{\text{phyFileName}}| \\
 &= 22 \times 11 \times 7 \times 5 \times 11 \times 9 \times 8 \times 9 \\
 &= 60,374,160.
 \end{aligned} \tag{2}$$

For the pure reinforcement-learning methods, the complete Cartesian product constitutes the actual global search space: combinations that violate action compatibility or current environmental prerequisites are not removed in advance, but remain selectable and receive failure penalties when executed. Consequently, only a small observation-dependent fraction of $\mathcal{A}$ can trigger valid state transitions. In L-ARLPT, the knowledge-driven layer generates a compact candidate action set C_t , with $|C_t| = K = 5$, and the DQN performs value evaluation only within this candidate set.

2.2.3. Reward Function Instantiation

In the proposed environment, the vast majority of parameter combinations fail to trigger valid state transitions, resulting in a severe scarcity of positive feedback signals. Consequently, the reward structure exhibits a highly sparse characteristic, where meaningful feedback is often obtained only after a sequence of prerequisite actions. To alleviate inefficient trial-and-error exploration and improve long-horizon credit assignment, existing cyber reinforcement learning studies have explored reward formulations that combine task-achievement rewards with intermediate action costs and progress-oriented guidance. Such reward shaping strategies help balance immediate efficiency and long-term objective optimization under sparse feedback conditions [28,29].

The reward design incorporates time-step cost, invalid-action penalty, micro-level state transition guidance, and macro-level milestone rewards. Importantly, successful

execution of an action does not always indicate progress toward the final penetration objective. Therefore, intermediate state changes are assigned micro-level rewards, while only strategically meaningful achievements receive macro-level milestone rewards. This hierarchical reward design enables the DQN agent to distinguish between locally successful transitions and globally valuable attack progress.

- **Time-Step Cost and Invalid-Action Penalty:** A time-consumption penalty ($r = -1$) is imposed at each interaction step to encourage convergence toward the shortest feasible attack path. Actions with incompatible parameters or unsuccessful execution outcomes receive an additional penalty ($r = -3$). Repeated operations that do not result in state transitions receive no reward ($r = 0$).
- **Micro-Level State Transition Guidance:** To mitigate the extreme sparsity of terminal rewards, progressive actions that successfully modify the local observation state, such as credential acquisition and privilege escalation, are assigned staged positive rewards ranging from ($r = +9$) to ($r = +13$). This mechanism is designed to assess the agent's ability to incrementally satisfy complex prerequisite dependencies through logical reasoning and sequential planning.
- **Macro-Level Milestone and Terminal Rewards:** A milestone reward ($r = +20$) is granted when the agent successfully compromises a new network node. Upon achieving the final objective within the core production domain, a substantial terminal reward ($r = +50$) is provided. This reward structure is intended to evaluate the agent's capability to overcome local optima and execute long-horizon strategic planning across multiple security domains.

Reward coefficients were empirically set via pilot experiments to preserve relative priorities among objectives, not to optimize absolute values.

3. Methodology

This section describes the proposed L-ARLPT framework, which integrates a knowledge-driven LLM layer with an experience-driven DQN layer for efficient penetration testing in large action spaces.

3.1. Framework Overview

To address the exploration inefficiency caused by high-dimensional combinatorial action spaces and sparse rewards in automated penetration testing, this paper proposes a Large Language Model-enhanced Reinforcement Learning framework, termed L-ARLPT. As illustrated in Figure 2, the proposed framework consists of two complementary modules: a knowledge-driven layer and an experience-driven layer. The knowledge-driven layer employs LLM-based semantic reasoning to prune the original action space into a compact candidate action set, while the experience-driven layer utilizes DQN with Prioritized Experience Replay (PER) to estimate action values and optimize long-term policies within the reduced space.

Forward Decision Flow (Environment $\rightarrow$ Action Execution): The knowledge-driven layer generates a candidate action set based on semantic reasoning, after which the DQN agent evaluates candidate actions and selects one for execution. The resulting reward and next observation are returned to update the policy and provide feedback for subsequent decisions.

Backward Learning Flow (Environment Feedback $\rightarrow$ Context Optimization): The feedback is interpreted jointly with strategic objectives, where successful state transitions only indicate environmental progress and do not necessarily imply strategic advancement. Since the LLM parameters remain frozen during training, execution outcomes are maintained through a Positive–Negative Separated Feedback Memory Mechanism and incorporated

into subsequent prompts through in-context feedback. This enables the LLM to refine candidate action generation based on environmental experience without modifying model parameters.

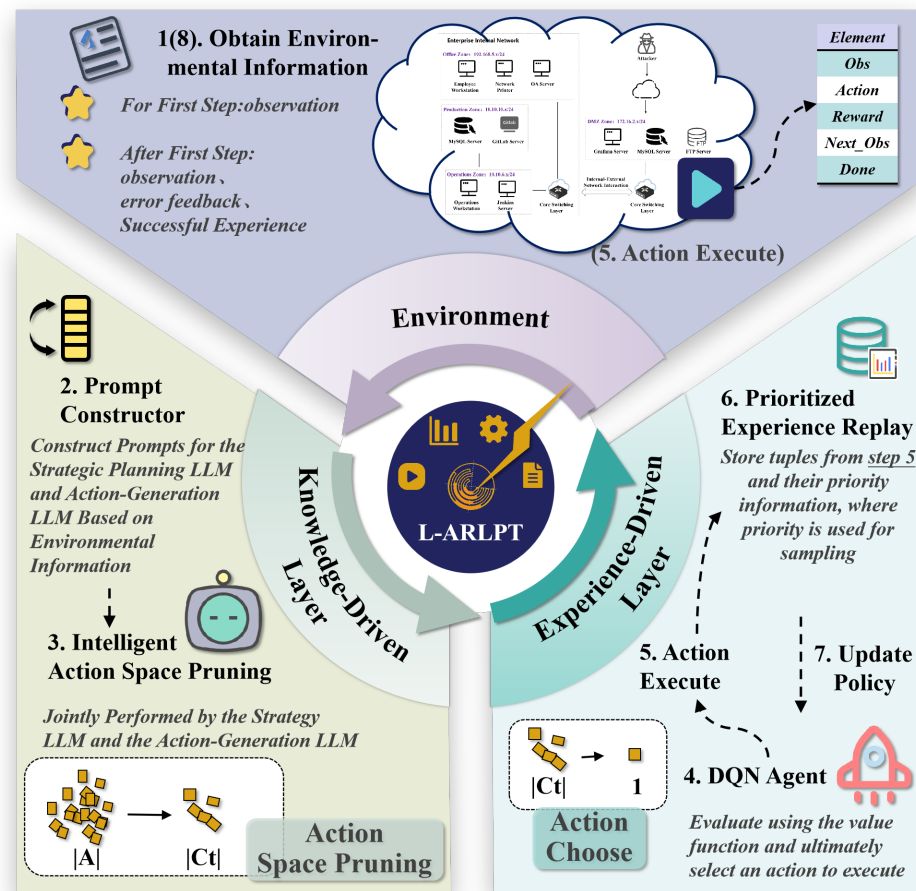


Figure 2. Overview of the L-ARLPT framework.

3.2. Knowledge-Driven Layer

The knowledge-driven layer employs two collaborative LLMs with different responsibilities: strategic planning and tactical action generation. As illustrated in Figure 3, the Strategic Planning LLM analyzes the current environment and historical feedback to produce prioritized targets, attack-stage assessments, and action-category guidance. The Action Generation LLM then combines these strategic instructions with current observations and execution feedback to instantiate parameterized candidate actions. Execution outcomes are incorporated into the next reasoning cycle, enabling iterative refinement without updating the LLM parameters.

Operationally, the two LLM components are invoked at different frequencies. The Strategic Planning LLM is invoked once at the beginning of each episode to establish the initial high-level strategy and is subsequently refreshed only when the preceding action produces a positive reward indicating state progress and the episode has not yet terminated. In contrast, the Action Generation LLM is invoked once at every decision step to construct the state-dependent candidate action set C_t . Therefore, a full 500-step episode involves up to 500 logical invocations of the Action Generation LLM, whereas a successfully terminated episode involves fewer calls according to its actual number of decision steps. The number of Strategic Planning LLM invocations is not fixed because it depends on the number of positive-progress transitions encountered during the episode. These counts refer to logical

model invocations; timeout, parsing, or invalid-response recovery may result in additional backend attempts under the retry mechanism.

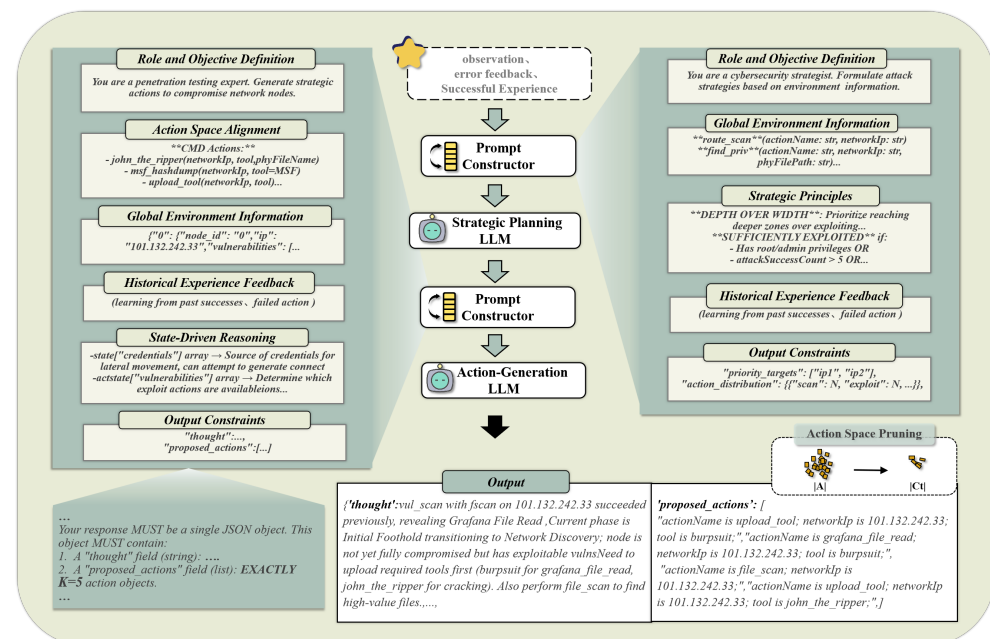


Figure 3. Hierarchical collaboration workflow of dual LLMs, Ellipses save space and indicate structure only.

To support the asymmetric functional division between strategic planning and tactical action generation, L-ARLPT employs Qwen3-14B as the Strategic Planning LLM and Qwen3-Max as the Action Generation LLM [30]. This configuration reflects the different requirements for efficient high-level reasoning and reliable constrained action generation, particularly because the latter is more sensitive to instruction deviation and invalid action fabrication. The detailed model-selection rationale and implementation settings are provided in Section 4.1.3.

3.2.1. Strategic Planning LLM: Macro-Level Planning and Coarse-Grained Action Pruning

The primary objective of the Strategic Planning LLM is to formulate attack strategies and perform the first-stage reduction of the decision space. To achieve this objective, the prompt is structured into five components: role and objective definition, global environment information, strategic principles, historical experience feedback, and output constraints. Within this framework, two key mechanisms are employed to achieve preliminary action-space reduction and hallucination mitigation.

Heuristic Target Constraints Guided by Strategic Principles: When processing complex network information, the Strategic Principles module incorporates expert-defined heuristic rules that guide the model to prioritize high-value assets located in deeper network segments and to assess target priority according to the current level of compromise. These heuristics impose constraints on both the target IP parameter and the selection of action categories within the action space. As a result, the search space is reduced at the order-of-magnitude level, preventing redundant exploration of already heavily exploited nodes and improving overall exploration efficiency.

Positive–Negative Separated Feedback Memory Mechanism: To reduce factual hallucinations, the Historical Experience Feedback module maintains previously executed actions and their corresponding outcomes as factual references for subsequent LLM reasoning. Positive feedback retains successful actions throughout the episode as a validated attack trajectory, while their environmental consequences are represented in the updated

observation rather than redundantly stored. Negative feedback maintains only the five most recent failed actions and is cleared after a successful state transition. This asymmetric memory design preserves validated attack trajectories while limiting the accumulation of unsuccessful exploration history. Leveraging the in-context learning capability of LLMs, this mechanism enables tactical adaptation based on environmental feedback without modifying model parameters, while reducing contextual noise and preventing repetitive trial-and-error exploration during long-horizon attacks.

Following the above reasoning process, the Strategic Planning LLM generates a standardized strategic instruction in accordance with the predefined output constraints. The instruction contains two key elements: a prioritized target IP list, which specifies the network nodes that should be preferentially attacked. An action-category allocation quota, which defines the maximum number of candidate actions to be generated for each action category. These outputs are subsequently incorporated into the prompt of the Action Generation LLM to provide high-level strategic guidance for candidate action generation.

3.2.2. Action Generation LLM: Tactical Instantiation and Candidate Action Generation

After the Strategic Planning LLM has established the high-level attack objectives, the Action Generation LLM receives the resulting strategic constraints and focuses on the second-stage reduction of the action space. Similar to the previous layer, its prompt follows a structured design comprising role definition, action-space alignment, environment information injection, historical experience feedback, state-driven reasoning rules, and output constraints. Within this framework, three key mechanisms are introduced to improve candidate action generation quality and action-space reduction efficiency.

Action Templates and Parameter-Space Alignment: The prompt explicitly specifies the action interfaces supported by the environment and defines the parameter templates associated with each action category. In addition, the permissible parameter values are strictly constrained to entities observed from the environment or entries contained in the predefined attack toolkit, such as common weak-password dictionaries and frequently used security tools. By explicitly restricting the action-generation space, the framework substantially reduces syntactically invalid or environmentally unsupported candidate actions.

State Verification and Attack-Condition Mapping: Guided by the state-driven reasoning rules, the LLM first analyzes the current environmental observations and then establishes mappings between discovered asset characteristics and appropriate attack-action categories. For example, vulnerability-related assets are mapped to exploitation-oriented attack actions, whereas credential-related assets are mapped to lateral-movement actions. This heuristic mapping mechanism enables the model to rapidly focus on high-value attack opportunities, thereby improving the quality of the generated candidate action set.

Action Validation: During candidate action generation, the Action Generation LLM is explicitly instructed to perform an internal validation step before finalizing each candidate action. Specifically, the model evaluates whether the generated action is consistent with the current environmental context and predefined execution constraints. At the network-topology level, the model checks whether the target IP is reachable and whether prerequisite dependencies, such as proxy-tunnel establishment for deep-network access, are satisfied. At the asset and execution levels, the model verifies the consistency between action parameters and the current environment state, including whether required resources, such as attack tools for payload execution, have been deployed to the target host. This validation process is implemented through structured prompting and does not involve an additional external validation module.

Finally, according to the predefined output constraints, the layer generates a standardized candidate-action object. The output consists of an explicit reasoning trace describing

the decision process and a state-dependent candidate action set C_t satisfying $|C_t| = K$, where K is configurable through the prompt.

Overall, the Strategic Planning LLM outputs prioritized targets and action-category quotas, which guide parameter alignment, asset-to-action mapping, and validation by the Action Generation LLM. This cascaded process reduces the original action space $|A|$ to the state-dependent candidate set C_t , allowing the DQN to focus on long-term value optimization among strategically relevant actions.

3.3. Experience-Driven Layer

The experience-driven layer employs DQN [31] with Prioritized Experience Replay [32] to optimize long-horizon attack decisions within the dynamically generated candidate action set. Unlike a fixed reduced action space, the candidate set is regenerated according to the current observation, strategic planning results, and environmental feedback, providing a state-dependent feasible action region for value optimization.

The experience-driven layer addresses two fundamental challenges. The first is to transform heterogeneous high-dimensional environmental observations and combinatorial actions into a unified tensor representation that can be processed by neural networks. The second is to establish a stable and efficient value-network evaluation and update mechanism while maintaining an appropriate balance between exploration and exploitation. The remainder of this section elaborates on these two core mechanisms in detail.

Observation Encoding: Penetration testing observations are inherently node-specific (e.g., vulnerabilities reside on hosts, and files are associated with user directories). A node-wise encoding causes state dimensionality to grow with the number of nodes, while heterogeneous node resources introduce invalid placeholders and highly sparse state representations. To address this issue, L-ARLPT adopts a global indexed-set encoding, where a global dictionary maps resource types from different nodes into a unified index space. Therefore, each state dimension represents whether a specific resource has been discovered in the current environment rather than a fixed attribute of a particular node. The corresponding state features and their design are presented in Table 1. This representation improves semantic consistency and enables the agent to learn transferable attack patterns across different network entities. Although this compression removes explicit resource–node ownership information from the observation vector, the underlying relationships are not discarded from the decision process.

3.3.1. Reinforcement Learning Observation and Action Encoding

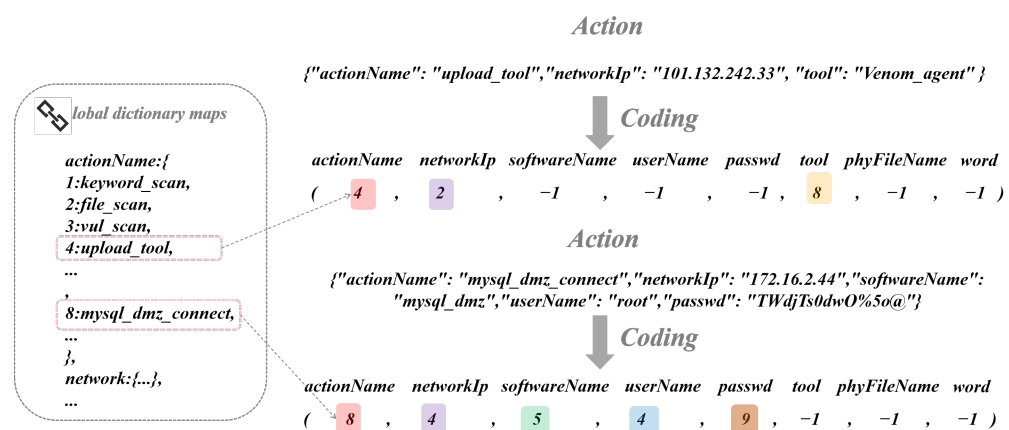
From the perspective of reinforcement learning representation, the penetration testing environment presents significant challenges due to its high-dimensional structured observations and complex combinatorial action space.

Under L-ARLPT's hierarchical architecture, this trade-off is addressed through the complementary roles of the knowledge-driven and experience-driven layers. The knowledge-driven layer preserves target-specific information through action parameters and environment-aware reasoning, while the experience-driven layer learns decision-relevant dependencies between observed resources, selected actions, and resulting attack outcomes through state-action-reward interactions. Therefore, the DQN does not need to explicitly reconstruct the complete topology ownership structure; instead, it learns value estimates based on resource availability and action consequences, improving policy generalization while maintaining effective attack decision-making.

Table 1. Attribute Design of the Reinforcement Learning Observation Vector.

Feature Group	Dimension	Design Objective	Encoding Method
Physical File Resources	11	Indicates whether critical physical files have been discovered	Binary encoding: set to 1 if the corresponding resource has been acquired by the attacker; otherwise 0
Software File Resources	5	Indicates whether software files have been discovered	
Physical Credential Resources	9	Represents acquired physical-user credential information	
Software Credential Resources	16	Represents acquired software-service access capabilities	
Discovered Node Vulnerabilities	11	Represents discovered exploitable vulnerability types	
Deployed Attack Tools	4	Indicates whether specific attack tools have been deployed to target nodes	
Discovered IP Resources	11	Represents the set of discovered network nodes	
Attack Failure Counter	1	Reflects the effectiveness of the current attack strategy	Normalized scalar value

Action Encoding: As illustrated in Figure 4, to address heterogeneous action representations, L-ARLPT adopts an eight-dimensional fixed-length action encoding based on predefined parameter slots. Instead of assigning an arbitrary numerical representation to each action, each dimension corresponds to a specific action-related attribute, such as action type, target entity, tool, vulnerability, credential, and other execution parameters. Therefore, different attack behaviors activate different semantic combinations of dimensions. For example, credential theft and vulnerability exploitation are represented by different action-type and parameter configurations rather than being mapped into indistinguishable vectors. Missing parameters are filled with default values to maintain a unified input dimension for the DQN while preserving the semantic structure of heterogeneous actions.

**Figure 4.** Action Encoding.

3.3.2. DQN-Based Decision Evaluation and Learning Mechanism

Online Decision-Making. After the knowledge-driven layer completes action-space pruning, it outputs a state-dependent candidate action set at each decision time step t :

$$C_t = \{a_t^{(1)}, a_t^{(2)}, \dots, a_t^{(K)}\}, \quad |C_t| = K \quad (3)$$

where K denotes the candidate-set size. Unlike conventional reinforcement learning methods that search over the entire action space A , the experience-driven layer performs value evaluation only within the current candidate action set C_t , thereby substantially reducing decision-making complexity. As illustrated in Figure 5, the online decision-making

process of the experience-driven layer consists of three stages: state–action representation construction, candidate-action value evaluation, and action selection. First, for the j -th candidate action $a_t^{(j)} \in C_t$, the current observation-state vector o_t is concatenated with the corresponding action-encoding vector to construct a unified state–action representation:

$$x_t^{(j)} = [o_t, a_t^{(j)}] \quad (4)$$

where o_t denotes the encoded environment observation, and $a_t^{(j)}$ represents the encoded candidate-action vector defined in Section 3.3.1. This joint representation simultaneously captures the current environmental state and the parameterized characteristics of the candidate action, providing the input features required for subsequent value estimation.

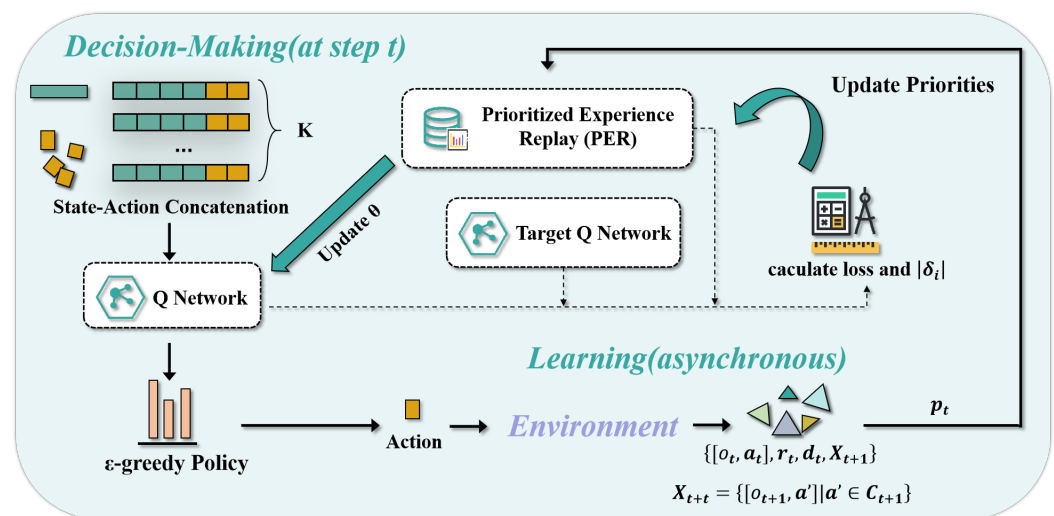


Figure 5. Workflow of Online Decision-Making and Prioritized Experience Replay in L-ARLPT.

Subsequently, the joint feature representation is fed into the Q-network to estimate the value of the corresponding state–action pair:

$$Q(o_t, a_t^{(j)}; \theta) \quad (5)$$

where θ denotes the parameters of the Q-network. This procedure is performed for all K candidate actions in C_t , yielding the following value set at the current decision step:

$$\left\{ Q(o_t, a_t^{(j)}; \theta) \right\}_{j=1}^K \quad (6)$$

Since the knowledge-driven layer has already imposed semantic constraints and prerequisite-condition filtering using cybersecurity domain knowledge, the experience-driven layer is relieved from evaluating a large number of semantically invalid actions and can instead focus its computational resources on comparing the long-term returns of the remaining candidate actions; the restriction is adaptive rather than static. The candidate set is continuously updated based on new observations, strategic objectives, and historical feedback, allowing potentially valuable actions to be introduced as the attack state evolves. In this stage, the Q-network serves to estimate the expected long-term cumulative return of alternative attack paths within the restricted candidate set, thereby compensating for the limited capability of large language models to explicitly quantify long-horizon decision value.

Action Selection. To balance exploration and exploitation, the proposed framework adopts the classical ϵ -greedy strategy during action selection. Specifically, with probability $1 - \epsilon$, the candidate action with the highest estimated value is selected:

$$a_t = \arg \max_{a \in C_t} Q(o_t, a; \theta) \quad (7)$$

With probability ϵ , an action is randomly sampled from C_t for exploration. Through this mechanism, the agent can effectively exploit accumulated experience while continuously exploring alternative high-value attack strategies generated by the knowledge-driven layer, thereby reducing the risk of premature convergence to suboptimal local optima.

The selected action is executed in the environment, yielding reward r_t , terminal indicator d_t , and next observation o_{t+1} . The replay buffer stores the replay entry $e_t = ([o_t, a_t], r_t, d_t, X_{t+1})$, where $X_{t+1} = \{[o_{t+1}, a'] \mid a' \in C_{t+1}\}$ contains the encoded features of the candidate action set generated online for the next observation.

DQN Learning Mechanism. As illustrated in Figure 5, during learning, the agent collects transitions and stores them in the replay buffer. Prioritized Experience Replay (PER) is used to improve the utilization of critical experiences under sparse rewards. For the i -th sampled transition, L-ARLPT constructs the Bellman target over its stored next-state candidate action set C_{i+1} . During replay updates, the encoded candidate features X_{i+1} are directly evaluated without invoking or updating the knowledge-driven layer. Accordingly, the target value is defined as:

$$y_i = r_i + \gamma(1 - d_i) \max_{a' \in C_{i+1}} Q_{\text{target}}(o_{i+1}, a'; \theta^-) \quad (8)$$

where C_{i+1} is the $K = 5$ candidate action set generated online for the next observation o_{i+1} , d_i is the terminal indicator, γ is the discount factor, and θ^- denotes the target-network parameters. The corresponding temporal-difference (TD) error is then computed as:

$$\delta_i = Q(o_i, a_i; \theta) - y_i \quad (9)$$

In PER, a priority value is assigned to each experience sample according to the magnitude of its TD error:

$$p_i = |\delta_i| + \eta \quad (10)$$

where η is a small positive constant introduced to prevent the sampling probability from becoming zero. The sampling probability of experience sample i is defined as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \quad (11)$$

where α controls the extent to which the assigned priority influences the sampling probability. To mitigate the estimation bias introduced by non-uniform sampling, an importance-sampling (IS) weight is incorporated:

$$w_i = \left(\frac{1}{NP(i)} \right)^\beta \quad (12)$$

where N denotes the capacity of the experience replay buffer, and β represents the bias-correction coefficient. Considering the substantial fluctuations of TD errors in the penetration-testing environment, Huber Loss is adopted as the optimization objective to improve training stability:

$$L(\theta) = \mathbb{E}[w_i \cdot \text{Huber}(Q(s_i, a_i; \theta) - y_i)] \quad (13)$$

$$\text{Huber}(x) = \begin{cases} \frac{1}{2}x^2, & |x| < \delta_{\text{huber}} \\ \delta_{\text{huber}} \left(|x| - \frac{1}{2}\delta_{\text{huber}} \right), & |x| \geq \delta_{\text{huber}} \end{cases} \quad (14)$$

Compared with the mean squared error (MSE) loss, Huber Loss provides quadratic optimization behavior for small TD errors and linear growth for large TD errors. Therefore, it preserves precise updates during normal learning while reducing the influence of occasional large TD errors caused by sparse rewards, invalid-action penalties, and exploration uncertainty, thereby improving training stability. To further alleviate target-value oscillation, a soft-update strategy is employed to maintain the target network:

$$\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^- \quad (15)$$

where τ denotes the soft-update coefficient. By combining the action-space constraints provided by the knowledge-driven layer with the long-horizon value-learning capability of PER-DQN, L-ARLPT is able to preserve the logical validity of candidate actions while further optimizing the global return of long-horizon attack paths, thereby enabling efficient exploration and policy optimization in high-dimensional combinatorial action spaces.

Although large language models may occasionally generate semantically plausible but environmentally invalid outputs, the proposed framework incorporates multiple mechanisms to limit their influence. The Action Generation LLM is constrained to instantiate actions only from a predefined action library with a fixed parameter schema, and candidate parameters are derived exclusively from environmental observations and knowledge-base entries. Furthermore, generated candidate actions are evaluated by the experience-driven layer before execution rather than being executed directly. If a selected action cannot be parsed or falls outside the global executable action space, the fallback mechanism replaces it with a valid action randomly sampled from the global action space. Meanwhile, the invalid output is recorded and incorporated into the subsequent prompt as negative feedback. Consequently, an isolated invalid generation neither interrupts the interaction loop nor directly terminates the ongoing attack path; its influence is confined to the current decision step, while the injected negative feedback helps reduce the recurrence of similar invalid outputs in subsequent reasoning cycles.

4. Results

This section presents experimental results evaluating the proposed framework against three research questions, covering learning efficiency, ablation contributions, and penetration capability.

- RQ1: Under penetration-testing scenarios with strict prerequisite dependencies and sparse rewards, how does L-ARLPT perform in terms of sample efficiency and convergence behavior compared with mainstream DRL algorithms and LLM-based baseline methods?
- RQ2: What are the respective contributions of prior-knowledge-driven pruning, the dual-LLM decoupled architecture, and the dynamic historical feedback mechanism to the overall decision-making performance of the framework?
- RQ3: Do the penetration-testing action sequences autonomously generated by L-ARLPT conform to the reasoning patterns and tactical progression expected by cybersecurity experts?

4.1. Experimental Setup

This subsection describes the simulation environment, task objective, evaluation metrics, and model configuration used in the experiments.

4.1.1. Simulation Environment and Task Objective

All experiments were conducted in the enterprise-grade high-fidelity simulation environment described in Section 2.2. At the beginning of each episode, the attacking agent was assumed to possess only the public IP address of the web portal located in the DMZ segment. The agent's objective was to autonomously perform a sequence of penetration-testing activities, including information gathering, vulnerability scanning and exploitation, privilege escalation, and lateral movement across internal networks, ultimately retrieving the target Flag from the protected asset located in the core production zone. To control interaction costs and emulate the constraints commonly encountered in real-world cybersecurity exercises, the maximum number of interaction steps per episode was limited to 500. An episode was considered successful if the agent obtained the target Flag within 500 steps, in which case the episode terminated immediately. Otherwise, if the step limit was reached before the objective was achieved, the episode was recorded as a failure.

4.1.2. Evaluation Metrics

To quantitatively evaluate the performance of different methods, three categories of evaluation metrics are employed.

- Global Evaluation of Learning Efficiency (RQ1: Comparative Performance Analysis):
 - Episode Cumulative Reward: Average cumulative reward per training episode. This metric captures exploration efficiency, sample convergence, and return optimization under sparse rewards, serving as a macro-level indicator of exploration stagnation or local-optima convergence.
- Mechanism-Oriented Evaluation of Core Framework Components (RQ2: Ablation Analysis):
 - Logical Violations and Invalid Prerequisite Actions: Average number of actions violating execution prerequisites in attack chains with strict precondition–postcondition dependencies. This metric evaluates the dual-LLM architecture's effectiveness in reducing contextual distraction and improving logical consistency during decision-making.
 - Positive and Negative Feedback Action Repetition Rate: Proportions of repeated failed and successful actions per episode, used to assess the effectiveness of the positive–negative separated feedback memory mechanism in suppressing factual hallucinations and reducing unnecessary trial-and-error exploration.
- Deep Penetration Evaluation for Practical Decision Quality (RQ3: Decision Behavior and Penetration Capability Analysis):
 - Action Behavior Distribution: During the stable convergence phase, executed actions are categorized into three groups: failed actions, redundant actions, and successful actions. This metric is used to analyze the exploration preferences of different methods when balancing penalty avoidance and long-term value maximization.
 - Average Penetration Depth per Episode: The simulated network is mapped into hierarchical penetration-depth levels according to its defense architecture. The deepest network zone reached through successful attack actions is recorded to quantitatively evaluate the agent's ability to traverse multiple security boundaries and perform long-horizon cross-domain penetration.
 - Average Decision Steps in Successful Episodes: This metric measures the average number of interaction steps required in episodes that successfully obtain the target Flag. Under the same network complexity, fewer decision steps indicate

lower trial-and-error costs, fewer redundant operations, and a more efficient attack chain.

4.1.3. Model Configuration and Implementation Details

All experiments were conducted using Python 3.9 and PyTorch 2.3.0 on an NVIDIA RTX A6000 GPU. In L-ARLPT, the knowledge-driven layer employs Qwen3-14B for strategic planning and Qwen3-Max for action generation. This selection follows the different functional requirements of the two LLM components. The Strategic Planning LLM mainly performs high-level penetration-context understanding, strategic target analysis, and coarse-grained action-space pruning, where a medium-scale locally deployed model provides a practical compromise between reasoning capability and inference overhead. In contrast, the Action Generation LLM is responsible for generating executable parameterized actions under environmental and formatting constraints, requiring stronger instruction-following and structured generation capability. Therefore, Qwen3-Max is adopted for this module to improve the reliability of candidate action generation. The default candidate-set size is set to $K = 5$, which provides a balance between preserving feasible attack diversity and maintaining efficient reinforcement learning optimization.

The reward function follows the formulation described in Section 2.2.3. The hyperparameters of L-ARLPT were determined through preliminary pilot experiments and fixed throughout all reported experiments. For baseline methods, algorithm-specific hyperparameters were selected to ensure stable convergence under the same environment (see Appendix A). Common training configurations were kept consistent across all methods. L-ARLPT and all comparison methods were independently executed ten times using the same set of ten seeds randomly sampled from $[0, 1000]$. In Figure 6, the solid lines denote the mean of the EMA-smoothed rewards ($\text{span} = 7$), while the inner and outer shaded regions denote ± 1 standard deviation and the minimum–maximum range, respectively.

The Strategic Planning LLM (Qwen3-14B) was deployed locally, whereas the Action Generation LLM (Qwen3-Max) was accessed through the official API. Following the invocation policy described in Section 3.2, a complete 500-step episode involved 500 logical Qwen3-Max invocations, while the number of strategic-planning calls depended on positive-progress transitions. Since provider-reported token metadata were not retained, token consumption was estimated from representative interaction logs. The estimated usage was approximately 4.75–5.43 thousand input tokens and 0.26–0.30 thousand output tokens per sampled interaction, corresponding to approximately 2.50–2.86 million Qwen3-Max tokens for a 500-step episode under a no-retry linear extrapolation. This value is intended only as an order-of-magnitude estimate because prompt length and occasional recovery retries vary during interaction. One complete independent L-ARLPT training run required approximately nine days. Per-call latency was not systematically recorded during the original experiments; therefore, reliable mean, median, and latency-variability statistics are unavailable.

4.2. Experimental Methodology

All competing methods were evaluated in the same simulation environment under an identical 500-step decision budget and reward configuration to ensure a fair comparison.

4.2.1. Baseline Methods

To address RQ1, four baseline methods from two categories were selected for comparison. The first category consists of mainstream deep reinforcement learning algorithms designed to handle large action spaces, aiming to evaluate the difficulty of learning solely through environment interaction in the absence of external domain knowledge. The second

category comprises LLM-based methods, which are used to assess the limitations of purely prior-knowledge-driven architectures in long-horizon decision-making tasks.

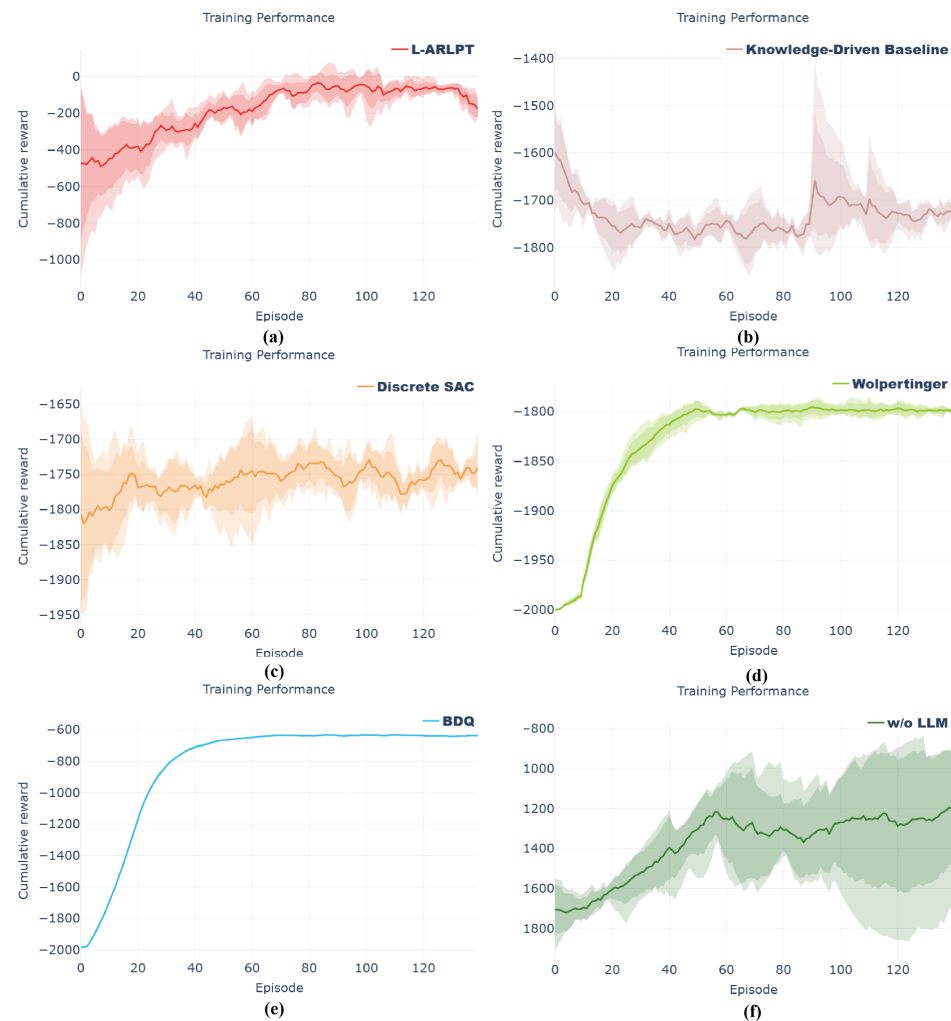


Figure 6. Episode Cumulative Reward Convergence Curves (a–f).

The selected reinforcement learning baselines were chosen because they represent three representative solution paradigms for large discrete action spaces, which directly correspond to the challenges posed by the parameterized combinatorial action space in autonomous penetration testing.

- BDQ (Branching Dueling Q-Network) [14]. A reinforcement learning architecture specifically designed for high-dimensional discrete action spaces. This method decomposes a combinatorial action space into independent branches according to action dimensions. Each branch estimates the advantage value of a sub-action, which is then combined with a shared state-value function to approximate the global Q-value.
- Wolpertinger [10]. A reinforcement learning architecture for large action spaces that maps discrete actions into a continuous embedding space. The model first generates an action vector in the continuous space and subsequently applies a k-nearest neighbor (k-NN) search within the embedding space to retrieve corresponding discrete actions.
- Discrete SAC [33]. A model-free discrete reinforcement learning algorithm based on the maximum-entropy framework, known for its strong exploration capability. To accommodate the multi-parameter action representation used in penetration testing, the actor network was modified into a branching structure that outputs independent categorical distributions for each action dimension.

- **Knowledge-Driven Baseline:** Uses only the knowledge-driven layer of L-ARLPT as the decision module. The first candidate action generated by the Action generation LLM (prompted with observations and historical feedback) is directly executed, removing DQN-based value evaluation and policy optimization.

4.2.2. Ablation Study Design

To address RQ2 and investigate the individual contributions of the core components within the L-ARLPT framework, three ablation variants were designed.

1. *w/o* LLM (Random Pruning + DQN). The knowledge-driven layer is replaced by uniform random sampling of k actions from the original action space, which are then evaluated by DQN. This design tests whether performance gains stem solely from reducing the candidate set size or from the LLM's context-aware logical filtering.
2. *w/o* Dual-LLM (Single-LLM + DQN Variant). Removes the Strategic Planning LLM, retaining only the Action generation LLM to process observations and generate candidate actions.
3. *w/o* Dynamic Feedback (No Historical Feedback Variant). Removes dynamic historical feedback from LLM prompts, designed to assess the positive–negative separated feedback memory mechanism in mitigating hallucinations and suppressing invalid actions.
4. **Sensitivity Analysis of Candidate Action Set Size.** The effectiveness of L-ARLPT relies on the interaction between knowledge-driven action pruning and experience-driven value optimization. An important question is whether the performance improvement originates from simply providing more candidate actions to the DQN or from constructing a compact and informative candidate space. Therefore, we investigate the influence of candidate action set size K . To isolate the influence of candidate-set size, we fix the trained DQN under the default configuration and vary only K during evaluation. Three settings ($K = 3$, $K = 5$, and $K = 10$) are compared, while keeping the LLM models, prompt templates, reward function, and evaluation environment unchanged.

4.3. Experimental Results and Analysis

This subsection presents the comparative evaluation, ablation study results, and decision behavior analysis.

4.3.1. Comparative Evaluation (RQ1)

The convergence curves of the average cumulative reward over 120 training episodes for different baselines are illustrated in Figure 6a–e. Based on these curves, the performance of each method is analyzed from three perspectives: sample efficiency, exploration behavior, and decision-making paradigm.

1. **Overall convergence behavior and sample efficiency.**
During the first 20 episodes, L-ARLPT's cumulative reward improves from approximately -500 to -400 , and the policy converges toward near-zero reward after about Episode 80. With step penalty -1 and failure penalty -3 , this indicates that the agent learns successful long-horizon attack paths while reducing ineffective actions and redundant interactions. The cross-run variability bands in Figure 6a are wide during early exploration but narrow after about Episode 60, indicating increasingly consistent performance across the ten runs rather than reliance on occasional successes. This suggests that LLM-guided pruning accelerates exploration and improves policy stability by enabling optimization within a semantically valid candidate set.
2. **Exploration challenges of traditional DRL in penetration testing.**
Although both BDQ and Wolpertinger exhibit relatively stable convergence after the initial learning stage, their reward curves plateau at substantially lower reward lev-

els than L-ARLPT. In particular, BDQ shows an almost deterministic convergence trajectory with minimal variability across repeated runs, while Wolpertinger rapidly converges to a stable yet low-reward policy. These observations indicate that the two methods consistently converge toward suboptimal local optima rather than discovering effective long-horizon attack strategies. For Discrete SAC, while maximum entropy promotes diversity, effective actions become negligible in the exponentially large combinatorial space. The maximum entropy objective thus disperses exploration over a vast ineffective space, causing exploration paralysis under sparse rewards. Moreover, the convergence curve remains centered around a relatively stable low-reward level throughout training without showing a clear upward trend, suggesting that entropy-driven exploration alone cannot effectively overcome the sparse-reward challenge in the exponentially large combinatorial action space.

3. **Limitations of Knowledge-Driven Models in Long-Horizon Decision Making.** The Knowledge-Driven Baseline consistently underperforms L-ARLPT throughout training. As shown in Figure 6b, a temporary increase in cumulative reward is observed around Episode 90, accompanied by noticeably wider cross-run variability bands. However, this improvement is not sustained in subsequent episodes, indicating that although the LLM occasionally generates effective attack sequences, the absence of reinforcement learning prevents these successful experiences from being consistently reproduced. Consequently, under dynamic network environments, the model remains prone to short-sighted decision loops guided solely by prior knowledge, resulting in inferior long-horizon planning capability.

4.3.2. Ablation Study Analysis (RQ2)

To quantify the individual contributions of the core components within the L-ARLPT framework, this section conducts an ablation study by jointly analyzing online training performance and offline generation-quality statistics.

1. **Effectiveness of Prior-Knowledge Pruning.** For the *w/o* LLM variant, Figure 6f shows a characteristic learning pattern over approximately 140 training episodes. The cumulative reward increases slowly early on, then stagnates with pronounced variance fluctuations later. The initial improvement indicates that DQN can acquire basic reconnaissance and exploration capabilities through interaction. The figure shows that although the cumulative reward gradually improves during the early training stage, the learning process becomes increasingly unstable after approximately Episode 60. The widening cross-run variability band indicates considerable cross-seed variability, suggesting that successful trajectories found through random exploration cannot be consistently reproduced. Thus, merely reducing the candidate-set size without semantic pruning is insufficient for stable long-horizon policy optimization.
2. **Necessity of the Dual-LLM Architecture.** The dual-LLM architecture was evaluated against the *w/o* Dual-LLM variant using action chains with strict prerequisite dependencies. In the representative sequence considered here, the SSH private key must first be obtained from the Grafana server before authentication-based access can be attempted. Before the key was acquired, the *w/o* Dual-LLM variant generated an average of 15.8 prerequisite-violating authentication attempts per episode, compared with 4.85 for L-ARLPT. This reduction indicates that separating high-level objective decomposition and prerequisite reasoning from low-level action instantiation improves logical consistency and reduces the burden on a single LLM to perform both functions simultaneously. Although these values are descriptive averages without formal statistical significance testing, they provide

empirical support for the dual-LLM design; more extensive repeated evaluations will be considered in future work.

3. Positive–Negative Separated Feedback Memory Mechanism.

The *w/o* Dynamic Feedback variant produced approximately 35.56% repeated failed actions and 3.53% repeated successful actions (re-exploitation of an already compromised host). L-ARLPT reduced these proportions to 23.15% and 0.94%, respectively. These reductions demonstrate that the mechanism effectively prevents wasted computation on redundant attempts. Although these behavioral statistics do not constitute a standalone hallucination rate, together with the prerequisite-violation analysis above, they provide task-level evidence that L-ARLPT reduces logically inconsistent and repeatedly ineffective LLM-generated suggestions.

4. Sensitivity Analysis of Candidate Action Set Size.

The candidate action set size K determines the degree of action-space reduction performed by the knowledge-driven layer and directly affects the subsequent DQN optimization. Therefore, we investigate whether the effectiveness of L-ARLPT benefits from an appropriate pruning granularity rather than simply increasing the number of generated actions.

The results are averaged over five evaluation episodes. As shown in Table 2, $K = 5$ achieves the best performance in terms of episode reward and penetration depth. Increasing K from 3 to 5 improves performance because the additional candidates provide better coverage of feasible attack strategies. However, further increasing K to 10 does not bring further improvement and instead decreases performance under the evaluated setting. This suggests that excessively enlarging the candidate space may reduce the benefit of semantic pruning by introducing additional low-value candidates and increasing the difficulty of value discrimination.

Table 2. Performance comparison under different K .

K	Episode Reward	APD
3	61	3.0
5	114	3.2
10	40	3.0

4.3.3. Decision Behavior Characteristics and Deep Penetration Capability Analysis

To characterize the decision-making behavior of the compared baselines, we analyzed interaction logs from the stable training phase. Executed actions were categorized into three groups: failed actions, redundant actions (repeated executions causing no state transition), and successful actions. Although the benchmark contains ten nodes, it should not be interpreted as a flat ten-host topology. The nodes are separated into four penetration-depth levels, and progression to a deeper level requires the agent to satisfy cross-zone reachability and attack-stage prerequisites rather than merely compromise additional hosts. To quantitatively evaluate the agents' ability to penetrate multiple security boundaries and achieve deep attack progression, we introduce the Average Penetration Depth (APD) metric. Logical security zones are mapped to depth levels according to their defense hierarchy: DMZ = 1.00, Office = 2.00, Operations = 3.00, and Core Production = 4.00. The penetration depth of a single episode is defined as the deepest security zone reached via successful execution of effective attack actions. The APD metric is calculated as follows:

$$APD = \frac{1}{N} \sum_{i=1}^N D_i \quad (16)$$

where N denotes the total number of evaluation episodes ($N = 40$), and D_i represents the maximum penetration depth achieved in the i -th episode. In Table 3, all statistics were computed over the final 40 episodes after training convergence. The reported values correspond to the average percentages of failed actions, redundant actions, and successful actions, the average number of decision steps per successful episode, and the Average Penetration Depth (APD), respectively.

Table 3. Decision behavior statistics and penetration performance comparison.

Method	Failed %	Redundant %	Successful %	Average Steps	APD
BDQ	10.04%	89.76%	0.20%	500	1
Wolpertinger	89.50%	9.71%	0.79%	500	1
Discrete-SAC	92.78%	5.82%	1.40%	500	1.27
Knowledge-Driven Baseline	92.77%	5.82%	1.42%	182	1.12
L-ARLPT	74.33%	19.02%	6.66%	128.43	3.33

1. Discrete SAC and Wolpertinger exhibit high failed-action rates of 92.78% and 89.50%, yielding APD values of only 1.27 and 1.00. Without prior constraints in the massive discrete action space, the agents rely primarily on random exploration. Under highly sparse rewards, actions that induce state transitions are rarely sampled, preventing the value network from accumulating sufficient positive experiences to support policy convergence. Consequently, exploration remains largely confined to the entry-level DMZ zone.
2. BDQ exhibits a distinctive action-distribution pattern, with failed actions accounting for only 10.04%, while redundant actions constitute 89.76% of all executed actions. Its APD remains at 1.00. Because failed actions incur explicit penalties, the BDQ agent tends to favor low-risk redundant actions that yield no state progression when its multi-branch exploration mechanism fails to converge toward a valid attack path. This behavior effectively minimizes punishment but prevents meaningful penetration progress.
3. The Knowledge-Driven Baseline records a failed-action rate of 92.77% and an APD of 1.12. The limitation arises from the lack of long-term cumulative return estimation, forcing exclusive reliance on greedy single-step reasoning from current observations. In long-horizon penetration scenarios, such myopic planning accumulates decision errors, frequently generating actions that violate reachability or prerequisite constraints and thus degrading policy stability.

In contrast, by combining LLM-guided prior-knowledge pruning with RL-based value evaluation, L-ARLPT achieves an APD of 3.33 with an average of 128.43 decision steps per successful episode. This indicates that the agent routinely penetrates beyond DMZ and Office zones, reaching the third-layer Operations zone and occasionally the Core Production zone. The improvement stems from the framework's ability to reshape the action distribution. The Knowledge-Driven Layer performs context-aware filtering on the high-dimensional action space, eliminating numerous logic-violating actions and reducing the failed-action proportion by approximately 15 percentage points compared to Wolpertinger and 18 percentage points compared to Discrete SAC and the Knowledge-Driven Baseline. Simultaneously, the Experience-Driven Layer explicitly estimates long-term returns via value approximation, raising the successful-action proportion to 6.66% while keeping redundant actions at 19.02%. These findings demonstrate that integrating commonsense reasoning with value-based policy optimization substantially enhances search efficiency and multi-boundary deep-penetration capability in long-horizon decision tasks.

To verify whether the decision trajectories generated by L-ARLPT conform to the reasoning paradigm typically adopted by human security experts, we analyze a representative successful episode that achieved the final objective within 151 decision steps.

1. **State Alignment and Prerequisite Unlocking (Initial Foothold Establishment):** During Steps 3–13 targeting the DMZ, the agent exploited a Grafana vulnerability to obtain an SSH key, used an offline cracking tool to recover credentials, then performed privilege escalation and deployed a proxy tunnel. This sequence follows vulnerability exploitation → credential acquisition → privilege escalation → pivot establishment, demonstrating coherent prerequisite-aware reasoning.
2. **Value-Driven Cross-Domain Lateral Movement (Credential Reuse):** In the lateral movement stage, the agent compromised an Office host (Step 22) and extracted plaintext credentials. While reconnoitering the Operations zone, it directly reused these credentials to establish an SMB connection (Step 48), a credential-reuse attack that reflects the model's ability to leverage intelligence gathered from prior stages.
3. **Objective Achievement:** Using recycle-bin log clues from an Operations host, the agent inferred the core code server's location and credentials, then exploited a GitLab remote code execution vulnerability to obtain privileged access and retrieve the Flag.

Overall, L-ARLPT autonomously constructs long-horizon attack chains that adhere to penetration-testing logic and causal attack dependencies.

5. Conclusions and Future Work

5.1. Summary and Limitations

This paper proposed L-ARLPT, an LLM-augmented reinforcement learning framework for autonomous penetration testing in environments characterized by high-dimensional combinatorial action spaces and sparse rewards. The proposed framework integrates a knowledge-driven layer and an experience-driven layer. The former leverages dual LLMs to perform hierarchical action-space pruning based on cybersecurity knowledge and environmental feedback, while the latter employs DQN with Prioritized Experience Replay to optimize long-horizon attack decisions within the reduced candidate action set.

Experimental results demonstrate that L-ARLPT improves exploration efficiency and attack-path discovery capability compared with representative reinforcement learning and LLM-based baselines. These results indicate that combining LLM-based semantic reasoning with reinforcement learning-based value optimization provides an effective approach for reducing ineffective exploration in autonomous penetration testing scenarios.

Despite the effectiveness demonstrated in the current evaluation, several directions remain open for further investigation. First, the current environment models state transitions deterministically to ensure controlled and reproducible evaluation, while future studies can investigate stochastic transition mechanisms to better characterize dynamic enterprise networks. Second, broader experimental studies with larger-scale repeated evaluations can further analyze the statistical robustness and general applicability of the proposed framework. Third, although the ten-node, four-zone benchmark is sufficient to reproduce the cross-domain causal structure targeted in this study, it does not by itself establish scalability with respect to increasing node count, topology heterogeneity, or the number of concurrent attack paths. Extending the simulation environment with more diverse network structures, vulnerability distributions, and adaptive defensive behaviors will provide additional insights into the scalability of autonomous penetration-testing agents.

In addition, the current study evaluates LLM reliability primarily through task-grounded behavioral evidence rather than an independently annotated candidate-level hallucination rate. Establishing a finer-grained evaluation protocol that distinguishes

schema violations, unsupported parameter generation, and prerequisite inconsistencies remains an important direction for future work.

5.2. Future Directions

Based on the above limitations, several directions will be investigated in future work. First, the simulation environment will be extended toward larger-scale and more dynamic enterprise networks, including stochastic state transitions and adaptive defense mechanisms, to further evaluate the robustness and generalization capability of L-ARLPT.

Second, future research will explore adaptive knowledge-driven mechanisms, including parameter-efficient fine-tuning, continual learning, and retrieval-based knowledge updates, allowing LLM components to incorporate newly observed environmental information while maintaining training stability.

Third, more systematic optimization strategies for reward design will be investigated, including automated reward adjustment and sensitivity analysis, to reduce dependence on manually defined reward parameters.

Finally, future evaluations will incorporate more extensive statistical validation through repeated independent experiments, enabling more rigorous analysis of performance variance and significance. These improvements will further enhance the reliability and applicability of L-ARLPT in practical autonomous penetration-testing scenarios.

Author Contributions: Conceptualization, R.Z. and L.W.; methodology, R.Z. and L.W.; software, R.Z.; validation, R.Z., J.Z., R.H. and Y.X.; formal analysis, R.Z. and C.C.; investigation, R.Z.; resources, R.Z. and J.Z.; data curation, R.Z. and J.Z.; writing—original draft preparation, R.Z. and J.Z.; writing—review and editing, L.W., R.Z., C.C. and J.Z.; visualization, R.Z.; supervision, R.Z.; project administration, L.W.; funding acquisition, L.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the National Natural Science Foundation of China (Grant No. 62572136 and No. W2511073).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

The following additional configurations complement the hyperparameter settings summarized in Table A1 and provide the complete empirical setup used for L-ARLPT training. PER used $\alpha = 0.6$, $\beta = 0.4$, and $\eta = 10^{-6}$, with replacement sampling, maximum-normalized importance-sampling weights, and maximum-priority initialization for new transitions (1.0 when the buffer was empty). The discount factor was set to $\gamma = 0.95$, where the Huber threshold parameter δ_{huber} was set to 1.0. The knowledge-driven layer employed Qwen3-14B for strategic planning and Qwen3-Max for action generation, both with a temperature of 0.3; their maximum output lengths were 2048 and 4096 tokens, respectively. Each request allowed three attempts, with timeout limits of 180 s and 300 s. The action-generation prompt requested $K = 5$ proposals.

Table A1. DQN Algorithm Hyperparameter Settings.

Method	L-ARLPT	BDQ	Wolpertinger
Learning Rate	3×10^{-4}	3×10^{-5}	Actor: 3×10^{-4} Critic: 3×10^{-5}
Discount Factor	0.95	0.999	0.999
Replay Buffer Size	1×10^6	1×10^6	1×10^6
Batch Size	512	512	512
Target network update	every 100 steps	every 100 steps	Soft update $\tau = 0.005$
Exploration strategy	ϵ -greedy	ϵ -greedy	Adaptive
Exploration parameters details	Initial ϵ : 1.0	Initial ϵ : 1.0	Initial ϵ : 0.1
	Min ϵ : 0.1	Min ϵ : 0.1	Max ϵ : 0.9
	Decay factor: 0.995	Decay factor: 0.995	Min ϵ : 0.1
			Success trigger (high/low): 30%/5%

Appendix B

The actionName domain contains 22 supported actions. Each action is associated with a parameter template that specifies its required fields, together with an operational description and an action category. The templates below list only the parameters required by each action.

1. **keyword_scan** (networkIP, word, tool): Enumerate directories or files on a target host or service. [Probing and Scanning]
2. **file_scan** (networkIP): Read sensitive files on a target node to obtain credentials, configuration, or network topology. [Probing and Scanning]
3. **vul_scan** (networkIP, tool): Attack a specified network using a previously scanned node as a pivot. [Probing and Scanning]
4. **network_scan** (networkIP, tool): Collect information about a target network. [Probing and Scanning]
5. **port_scan** (networkIP, tool): Scan ports on a target host. [Probing and Scanning]
6. **grafana_file_read** (networkIP, tool): Exploit CVE-2021-43798 to read files from a Grafana server. [Exploitation]
7. **ms17_010** (networkIP, tool): Exploit EternalBlue (MS17-010) to gain system privileges. [Exploitation]
8. **oa_rce** (networkIP): Exploit CVE-2022-22916 to compromise an O2OA server. [Exploitation]
9. **jenkins_rce** (networkIP): Exploit CVE-2018-1000861 to compromise a Jenkins server. [Exploitation]
10. **gitlab_rce** (networkIP): Exploit CVE-2021-22205 to compromise a GitLab server. [Exploitation]
11. **weak_passwd** (networkIP, userName, passwd): Attempt to log in to a target server using weak passwords. [Exploitation]
12. **ifconfig** (networkIP, tool): Obtain network interface information of a target node. [Command Execution]
13. **upload_tool** (networkIP, tool): Upload an attack tool to a target node. [Command Execution]
14. **msf_hashdump** (networkIP, tool): Dump NTLM hashes from a Windows system. [Command Execution]
15. **john_the_ripper** (networkIP, tool, phyFileName): Offline cracking of hash files or SSH keys. [Command Execution]

16. **ftp_connect** (networkIP, userName): Log in to an FTP server using anonymous credentials (e.g., username “anonymous”). [Lateral Movement]
17. **mysql_dmz_connect** (networkIP, userName, passwd, softwareName): Connect to a MySQL database in the DMZ zone. [Lateral Movement]
18. **mysql_core_connect** (networkIP, userName, passwd, softwareName): Connect to a MySQL database in the core production zone. [Lateral Movement]
19. **oa_connect** (networkIP, userName, passwd, softwareName): Log in to the O2OA system administration backend. [Lateral Movement]
20. **web_connect** (networkIP, userName, passwd, softwareName): Log in to the GitLab administration backend. [Lateral Movement]
21. **smb_connect** (networkIP, userName, passwd, softwareName): Log in to a target Windows host via the SMB protocol. [Lateral Movement]
22. **sudo_vim_priv** (networkIP): Escalate privileges from a low-privileged user to root via a misconfigured sudo-vim. [Privilege Escalation]

References

1. Enoch, S.Y.; Huang, Z.; Moon, C.Y.; Lee, D.; Ahn, M.K.; Kim, D.S. HARMer: Cyber-Attacks Automation and Evaluation. *IEEE Access* **2020**, *8*, 129397–129414. [\[CrossRef\]](#)
2. Palmer, G.; Parry, C.; Harrold, D.J.; Willis, C. Deep reinforcement learning for autonomous cyber defence: A survey. *arXiv* **2023**, arXiv:2310.07745. [\[CrossRef\]](#)
3. Bahrami, P.N.; Dehghantanha, A.; Dargahi, T.; Parizi, R.M.; Javadi, H.H. Cyber kill chain-based taxonomy of advanced persistent threat actors: Analogy of tactics, techniques, and procedures. *J. Inf. Process. Syst.* **2019**, *15*, 865–889. [\[CrossRef\]](#)
4. Liu, J.; Zhang, Y.; Zhou, S.; Yang, J.; Lu, Y.; Zhong, X. Autonomous penetration testing using reinforcement learning: A review and perspectives. *Expert Syst. Appl.* **2025**, *300*, 130219. [\[CrossRef\]](#)
5. Alwabisi, S.O. AI in Penetration Testing: A Systematic Mapping Study. *TechRxiv* **2025**. [\[CrossRef\]](#) [\[PubMed\]](#)
6. Fernandes, R.; Lopes, N.; Gonçalves, J.; Cosgrove, J. Autonomous Pentesting Using Reinforcement Learning: A Systematic Literature Review. *SSRN* **2024**, 5208526. [\[CrossRef\]](#)
7. Simon, R.; Mees, W. SoK: A comparison of autonomous penetration testing agents. In Proceedings of the 19th International Conference on Availability, Reliability and Security, Vienna, Austria, 30 July–2 August 2024; pp. 1–10. [\[CrossRef\]](#)
8. Simon, R.; Libin, P.; Mees, W. Learning Robust Penetration-Testing Policies under Partial Observability: A systematic evaluation. *arXiv* **2025**, arXiv:2509.20008. [\[CrossRef\]](#)
9. Hong, S.; Yue, T.; You, Y.; Lv, Z.; Tang, X.; Hu, J.; Yin, H. A resilience recovery method for complex traffic network security based on trend forecasting. *Int. J. Intell. Syst.* **2025**, *2025*, 3715086. [\[CrossRef\]](#)
10. Dulac-Arnold, G.; Evans, R.; van Hasselt, H.; Sunehag, P.; Lillicrap, T.; Hunt, J.; Mann, T.; Weber, T.; Degris, T.; Coppin, B. Deep reinforcement learning in large discrete action spaces. *arXiv* **2015**, arXiv:1512.07679. [\[CrossRef\]](#)
11. Nguyen, H.V.; Nguyen, H.N.; Uehara, T. Multiple level action embedding for penetration testing. In Proceedings of the 4th International Conference on Future Networks and Distributed Systems, St. Petersburg, Russia, 26–27 November 2020; pp. 1–9. [\[CrossRef\]](#)
12. Zhou, S.; Liu, J.; Lu, Y.; Yang, J.; Hou, D.; Zhang, Y.; Hu, S. APRIL: Towards scalable and transferable autonomous penetration testing in large action space via action embedding. *IEEE Trans. Dependable Secur. Comput.* **2024**, *22*, 2443–2459. [\[CrossRef\]](#)
13. Tran, K.; Standen, M.; Kim, J.; Bowman, D.; Richer, T.; Akella, A.; Lin, C.T. Cascaded reinforcement learning agents for large action spaces in autonomous penetration testing. *Appl. Sci.* **2022**, *12*, 11265. [\[CrossRef\]](#)
14. Tavakoli, A.; Pardo, F.; Kormushev, P. Action branching architectures for deep reinforcement learning. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32. [\[CrossRef\]](#)
15. Lin, H.; Huang, C.; Chen, Z. An Advantage-based Optimization Method for Reinforcement Learning in Large Action Space. *arXiv* **2024**, arXiv:2412.12605. [\[CrossRef\]](#)
16. Ren, Q.; Liu, J.; Xiong, X. Automated penetration testing based on action clipping and decomposition. In Proceedings of the 2025 4th International Symposium on Computer Applications and Information Technology (ISCAIT), Xi'an, China, 21–23 March 2025; pp. 1740–1748. [\[CrossRef\]](#)
17. Hu, Z.; Beuran, R.; Tan, Y. Automated penetration testing using deep reinforcement learning. In Proceedings of the 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Virtual, 7–11 September 2020; pp. 2–10. [\[CrossRef\]](#)

18. Li, F.; Wang, X.; Liu, Y.; Luo, L. Penetration test path discovery based on NHSC-PPO. In Proceedings of the 2023 7th International Conference on Electronic Information Technology and Computer Engineering, Xiamen, China, 20–22 October 2023; pp. 1760–1765. [\[CrossRef\]](#)
19. Zahavy, T.; Haroush, M.; Merlis, N.; Mankowitz, D.J.; Mannor, S. Learn what not to learn: Action elimination with deep reinforcement learning. In Proceedings of the NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montréal, QC, Canada, 3–8 December 2018. [\[CrossRef\]](#)
20. Wang, P.; Liu, J.; Zhong, X.; Yang, G.; Zhou, S.; Zhang, Y. DUSC-DQN: An improved deep Q-network for intelligent penetration testing path design. In Proceedings of the 2022 7th International Conference on Computer and Communication Systems (ICCCS), Virtual, 22–25 April 2022; pp. 476–480. [\[CrossRef\]](#)
21. Wu, L.; Liu, J.; Yang, J.; Zhong, X. Automated Penetration Testing Through Hierarchical PPO with Large Language Model Enhancement. In *Proceedings of the 21st International Conference on Intelligent Computing, Ningbo, China, 26–29 July 2025*; Springer Nature: Singapore, 2025; pp. 472–486. [\[CrossRef\]](#)
22. Li, Y.; Dai, H.; Yan, J. Knowledge-informed auto-penetration testing based on reinforcement learning with reward machine. In Proceedings of the 2024 International Joint Conference on Neural Networks (IJCNN), Yokohama, Japan, 30 June–5 July 2024; pp. 1–9. [\[CrossRef\]](#)
23. Li, Q.; Wu, J. Efficient network attack path optimization method based on prior knowledge-based PPO algorithm. *Cybersecurity* **2025**, *8*, 15. [\[CrossRef\]](#)
24. Tucker, M.; Tuckute, G. Increasing brain-llm alignment via information-theoretic compression. In Proceedings of the UniReps: The First Workshop on Unifying Representations in Neural Models, New Orleans, LA, USA, 15 December 2023.
25. Wang, G.; Xie, Y.; Jiang, Y.; Mandlkar, A.; Xiao, C.; Zhu, Y.; Fan, L.; Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *arXiv* **2023**, arXiv:2305.16291. [\[CrossRef\]](#)
26. Ma, W.; Mi, Q.; Zeng, Y.; Yan, X.; Wu, Y.; Lin, R.; Zhang, H.; Wang, J. Large language models play starcraft ii: Benchmarks and a chain of summarization approach. *Adv. Neural Inf. Process. Syst.* **2024**, *37*, 133386–133442. [\[CrossRef\]](#)
27. Monahan, G.E. State of the art—A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Manag. Sci.* **1982**, *28*, 1–16. [\[CrossRef\]](#)
28. Shahid, A.; Ishum, I.M.; Haque, A.K.M.; Rahman, M.S.; Islam, A.B.M. Adversarial Reinforcement Learning for Offensive and Defensive Agents in a Simulated Zero-Sum Network Environment. *arXiv* **2025**, arXiv:2510.05157. [\[CrossRef\]](#)
29. Shaikh, A.J.; Bastian, N.D.; Shah, A. A red teaming framework for evaluating robustness of AI-enabled security orchestration, automation, and response systems. *arXiv* **2026**, arXiv:2605.17075. [\[CrossRef\]](#)
30. Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. Qwen3 technical report. *arXiv* **2025**, arXiv:2505.09388. [\[CrossRef\]](#)
31. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602. [\[CrossRef\]](#)
32. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized experience replay. *arXiv* **2015**, arXiv:1511.05952. [\[CrossRef\]](#)
33. Christodoulou, P. Soft actor-critic for discrete action settings. *arXiv* **2019**, arXiv:1910.07207. [\[CrossRef\]](#)

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.