ingegneria sismica
International Journal of Earthquake Engineering
Anno XLIII - Num. 2 - 2026

Automated Penetration Test Path Optimization and Decision Making Based on Artificial Intelligence Models

Wei Li[1], *Zixuan Zhao*[1,*], *Youchen Shi*[1], *Yinquan Wang*[1], *Zeyang Zhao*[1] *and Hanlin Tu*[1]

[1] Digital Intelligence Technology Company, PetroChina Xinjiang Oilfield Company, Karamay, Xinjiang, 834000, China

SUMMARY: *Automating penetration testing has been a challenge as it requires extensive expertise and experience from security professionals and usually requires a tedious manual testing process. In this study, with the help of an artificial intelligence model, a Markov decision process is constructed for describing the process of defining penetration tests. The objective is established to maximize the cumulative reward value and acquire an optimal strategy to guarantee the maximization of the expected return. Subsequently, the state and action reward functions are defined to accomplish penetration test modeling. A DQN algorithm founded on deep reinforcement learning is put forward to obtain the optimal strategy by learning the precise Q - function during environmental interactions. Moreover, a Dueling DQN algorithm known as empirical campaigning is proposed to more effectively handle the intricate state and action space.For the purpose of verifying the penetration success ratio of the arithmetic method this research employed, an aggressive person carries out an attack toward the object under the experimental environment. The attack measurement index of the penetration attack which this study carries out is comparatively high, reaching as high as 27.348. This effect exceeds the attack index values of the other two wide-used calculation methods.Comparing the optimal environmental reward values of the algorithms in different scenarios, the Dueling_DDQN algorithm is able to reach convergence in fewer training times, and reaches the desired reward value after 50 training times. It shows that the algorithm in this paper is able to achieve optimization and decision making for automated penetration test paths.*

KEYWORDS: *Artificial Intelligence Model; Markov Decision Making; Dueling_DDQN; Q-Function; Automated Penetration*

1 Introduction

With the continuous updating and iteration of related technologies in the field of computer, the utilization rate of computer network is also rising. But the computer network brings convenience to people's lives at the same time, computer network security problems also occur frequently [1]. Network security protection is often easy to attack and difficult to defend, the cost of defending against hacker attacks is much higher than the cost of acting as a hacker, because the defense of hackers need to find all the vulnerabilities of the system, while hackers only need to find a loophole can be attacked successfully to paralyze the system, so the corporate government had to invest a large sum of money in the researchers and the new defense technology [2-5].

*wlaqzxzzx@163.com
https://doi.org/10.65102/is2026574 Published: 30 April, 2026

With the development of the industry, people gradually realized that the research of security protection technology is not only consuming large sums of money, but also the effect is not significant, because the hackers can always find an attack method to bypass the protection strategy to attack [6-9]. So penetration testing gradually developed in the security field and has a vital role in the assessment of network security protection. Penetration testing is a network security testing method that exploits system vulnerabilities and gains control of the system by simulating hacker attacks without affecting the network of the target system [10]. Through the above method, the target system can take corresponding protective measures based on the vulnerabilities in the report, preventing them from happening in the first place, making the system network more robust and less susceptible to hacker attacks [11, 12].

However, the modern enterprise or organization's computer network system is large and complex, which runs a variety of software is still constantly updating, modifying, deleting, migrating and other changes, rely solely on security experts and security analysts to implement security testing is impractical [13-15]. The current emergence of penetration testing tools rely heavily on expert knowledge, and the whole set of testing process requires the use of a variety of different testing tools, requiring operators to have penetration testing related expertise in order to utilize the tools to implement the test [16-19]. In order to break through the above bottlenecks, artificial intelligence (AI) of medium and large language models and reinforcement learning algorithms provide new perspectives for penetration testing research.

Ren et al [20] (2024) proposed innovative proximal policy optimization algorithms for automated penetration testing by combining a long and short-term memory network architecture with an advanced curiosity exploration mechanism that enhances the utilization of historical empirical information and the exploration of unknown environments. Ren et al [21] (2024) created a new algorithm for task decomposition of multiple intelligences through long and short-term memory networks and multi-task decomposition methods, which achieved higher cumulative rewards with fewer steps, optimized decision-making capabilities in intelligent penetration testing, and outperformed traditional proximal policy optimization algorithms. Samad et al [22] (2024) designed an intelligent automated penetration testing system using AI techniques such as reinforcement learning, which can be seamlessly integrated with existing penetration testing frameworks to synchronize and optimize human resources and improve time efficiency, reliability, and testing frequency. Li et al [23] (2024) provide an automated penetration testing strategy for dynamic network scenarios, which achieves dynamic scenario capture through reinforcement learning algorithms and makes decisions based on historical experience, maintaining the penetration testing agent's learning flexibility and adaptability. Moreno et al [24] (2025) combined reinforcement learning with a context-rich recommender system with vocabulary recognition capabilities, where reinforcement learning evaluates the optimal attack strategy through the data while dynamically capturing alternative attack paths, obtaining higher than 97% accuracy and effectiveness in autonomous penetration test path selection. Sun et al [25] (2025) integrated prior knowledge and deep reinforcement learning to achieve intelligent probing of optimal attack paths for automated penetration testing in unknown environments and optimized the console dynamics problem by combining the attention mechanism to optimize the path decision for penetration testing of power IoT systems. Shen et al [26] (2025) utilized the advantages such as the power of type language models and retrieval-enhanced generation, combined with multi-intelligence collaboration, to construct an automated penetration testing framework, which effectively automated intelligence gathering, vulnerability analysis, and

various tasks. Deng et al [27] (2024) evaluated the performance of a penetration testing approach based on a large-scale language model, and only some of the sub-tasks demonstrated better performance, but there are still deficiencies in maintaining the overall context of the test scenario.

However, the current automated penetration testing techniques based on AI models still have limitations. For this reason, Wang et al. [28] (2025) designed a network penetration testing framework called “PTFusion”, which utilizes a large-scale language model and an intelligent fusion mechanism based on context-awareness to achieve more accurate strategic planning and execution of penetration testing commands. Alghamdi [29] (2025) pointed out that reinforcement learning, while it can be applied in automated penetration testing either directly or in conjunction with other techniques, suffers from multiple challenges such as high computational cost, the presence of reward sparsity, and barriers to practical application.

In this research, we use the Markov decision-making process to give the procedural definition of penetration testing. At the same time, the problem about immediate and long-term rewards is being considered by us, therefore, the goal of the intelligent agent is set up to be the optimization of accumulated rewards. One certain degree of randomness is been incorporated into the learning stage of the intelligence body therefore to make exploration of the environmental situations.Subsequently, the optimal strategy is adopted as the starting point to optimize the anticipated reward. The states and actions of the intelligent agents are defined independently, and the objective of penetration testing is established as gaining control over the computer's mainframe. A value-based DQN algorithm is proposed by combining Q-learning and deep neural network, and an improvement is proposed to build the Dueling DQN algorithm named empirical campaigning. The penetration test learning algorithm using Q-Learning and Dueling DQN is used to learn behavioral strategies during penetration test interactions and complete the optimization and decision-making of automated penetration test paths. Construct a simulation experimental environment in which the attacker attacks the target to verify the penetration success rate and attack value of the algorithm of this research, and run the algorithm in different scenarios to evaluate the reward value situation of the best environment to achieve the optimization and decision-making of the automated penetration test path.

2 Markov decision process-based modeling for penetration testing

2.1 Markov Decision Process

2.1.1 Process definition

Using a Markov decision process to describe the penetration test definition process can be defined as a quaternion $(\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{R})$, where the elements have the following meanings:

(1) $\mathcal{S}$, the set of states, also known as the state space. The $\mathcal{S}$ contains a finite number of states s. The state s is a description of the environment.

(2) $\mathcal{A}$, the set of actions, also known as the action space. The $\mathcal{A}$ contains a finite number of actions a. This aggregation includes all the actions which an intelligence-based body has the ability to carry out.

(3) $\mathcal{T}$, the state transfer function, $\mathcal{T}:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\mapsto\mathbb{R}$. This function specifies the probability that the environment will move from the current state s to the next state s' under the action a.

(4) $\mathcal{R}$, the reward function, $\mathcal{R}:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\mapsto\mathbb{R}$. This function specifies the reward that the environment feeds back to the intelligent body after it is transferred from the current state s to the next state s' under the action a.

Based on the above definition, the interaction between the intelligent body and the environment can be expressed as the sequence shown in equation (1). Where $S_0, S_1, \ldots \in \mathcal{S}$, $A_0, A_1, \ldots \in \mathcal{A}$, $R_t = \mathcal{R}\left(S_{t-1}, A_t, S_t\right)$. The intelligent body starts from an initial state S_0 where it perceives the environment and chooses an action A_0 based on a strategy. The environment shifts to state S_1 and feeds back reward R_1:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \cdots \tag{1}$$

The above sequence records the flow of interaction between the intelligent body and the environment, which may be represented in FIG. 1.

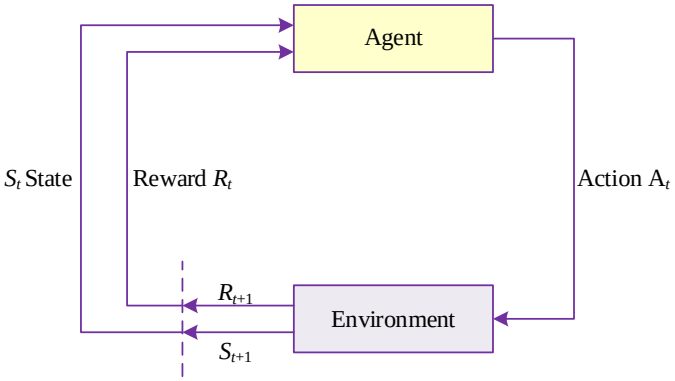


Figure 1: The interaction process between the agent and the environment

At moment t, the intelligent body senses the state S_t of the environment and chooses the action A_t, by which the environment is shifted from state S_t to S_{t+1}, while the environment outputs a reward R_{t+1}. This is a scalar value that is used to provide feedback on the merits of the action A_t.

In the sequence shown in (1), the state transfer function gives the probability of transition between different states of the environment. This demonstrates the ever - changing character of the environment, as well as the reality that the state changes of the environment adhere to the Markov property: the state of the environment at the subsequent moment, $S_{t+1} = s'$, depends only on the previous moment state, $S_t = s$, and the action, $A_t = a$, and is independent of the states and actions prior to the moment t, as shown in equation (2). Thus the state of the environment at moment t must include all information prior to moment t that has an impact on the future, which is generated by the interaction of the intelligent body with the environment:

$$p\left(S_{t+1}\middle|S_t, A_t, \ldots, S_0, A_0\right) = p\left(S_{t+1}\middle|S_t, A_t\right) \tag{2}$$

When the reinforcement learning task is a class of repetitive interactions, e.g., playing Go, then the process of an intelligent body interacting with the environment can be divided into multiple subsequences. Each subsequence is a finite sequence containing a start state and an

end state S_T, with T representing the final moment. Such a task is called a turn-based task. From the start state to the end state constitutes a turn. Penetration testing can be viewed as a turn-based task. A penetration tester focuses on one host during the penetration process, takes control of that host and then starts penetrating the next host. The process of going from having no knowledge of a host to having complete control over it can be considered a round in a penetration testing task.

2.1.2 Income and returns

The objective of an intelligent entity is to optimize the accumulated reward, which is also known as payoff. In a turn-based task, the sequence of rewards earned by an intelligent body after the t moment is shown in (3). Where T is the termination moment of a round:

$$R_t, R_{t+1}, R_{t+2}, R_{t+3}, \ldots, R_{T-1}, R_T \tag{3}$$

In order to maximize the reward, the intelligent body needs to consider not only maximizing the immediate reward in the current state, but also the possible reward in the future period when choosing an action, and needs to balance the relationship between the immediate reward and the future reward according to the actual task needs. In order to weigh the immediate reward and future reward, a discount factor is added to the reward. At this point, the reward obtained by the intelligent body at the moment t can be expressed by equation (4):

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-t-1} R_T = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{4}$$

where R_k is the immediate reward received by the intelligent at moment k and $k \in [t+1, T]$. γ is the discount factor and $\gamma \in [0,1]$. The discount factor determines the present value of the future reward, and the reward R_{t+n} at moment $t+n$ is discounted to the reward $\gamma^n R_{t+n}$ received at moment t. If γ is set to 0, the reward at moment $t+1$ is considered worthless at moment t and the reward is equivalent to the current reward. At this point, the intelligence only considers the immediate reward and needs to learn how to choose A_t in order to maximize the value of R_{t+1}. If the value of γ is less than 1,after that, the clever body considers not only the direct repayment but also therefore has to take into account the future award. However, due to the existence of γ, the intelligent body does not regard immediate rewards and future rewards as equally important. As the value of γ gets closer to 1, the intelligent body considers the future reward more when choosing the action A_t. When the value of γ is equal to 1, the intelligent body will regard immediate and future rewards as equally important.

2.1.3 Action strategies

Regardless of the value of γ, at any $t \in [0, T]$ moment, the intelligent body needs to choose the action A_t in the environment state S_t, and the basis on which the intelligent body chooses the action is called the strategy, denoted by π. There are two kinds of strategies: deterministic strategies and stochastic strategies. A deterministic policy is a mapping from the state space $\mathcal{S}$ to the action space $\mathcal{A}$, denoted as $a = \pi(s)$, where $a \in \mathcal{A}, s \in \mathcal{S}$. The

stochastic strategy represents the probability of the intelligent selecting each action a given the environmental state s, denoted by $\pi(a|s)$, and $\sum_a^{\mathcal{A}}\pi(a|s)=1$, where $s\in\mathcal{S},a\in\mathcal{A}$. Intelligent bodies generally use stochastic strategies to introduce some randomness in the learning process to better explore the environment. When both the strategy and the state transfer function are probabilistic, the subsequence generated by the interaction between the intelligent body and the environment in a turn-based task is a randomized sequence, and the intelligent body harvests a different reward G_0 at each turn. Therefore the intelligent body needs to learn an optimal strategy π^* to maximize the expected return.

2.2 Penetration Test Modeling Implementation

2.2.1 Status definitions

State is a description of the environment. In order to describe the penetration test environment, the state is defined by six parameters. In the previous section, it was pointed out that penetration testing is a round-robin task, so the process of penetrating a single host is considered in the modeling process.

2.2.2 Action definitions

The actions that can be used by the intelligence correspond to the operations that can be used during penetration testing, take one example: data gathering, fragility use, and after-breaking-in behaviors. Information gathering is realized by the Nmap tool, which supports probing the target host's operating system, open ports and service information.

2.2.3 Reward functions

Within a Markov decision - making process, the objective of an intelligent agent is to choose actions that optimize the accumulated rewards. Therefore, the reward signal must be capable of mirroring the aims of the real - world task. In this research, the target of penetration testing is defined as gaining control over the computer host. In this research paper, we set the objective that penetration testing is to obtain the control of the host computer. In this regard, the entire process of penetrating the host computer by an intelligent body is considered: information collection, vulnerability exploitation, elevation of privileges, and establishment of a backdoor on the target host computer in turn. Analyzing the process, the three steps of vulnerability exploitation, elevation of privilege and establishment of backdoor gradually increase the degree of control of the intelligent body over the target host. Consequently, the reward function within this model is presented in the equation (5):

$$\mathcal{R}=\begin{cases}-1, & \text{Cost of executing arbitrary actions}\\ 1, & \text{Establish a shell session}\\ 6, & \text{Establish a Meterpreter session}\\ 6, & \text{Gain administrator privileges}\\ 10, & \text{Create a persistent backdoor}\end{cases} \tag{5}$$

3 Reinforcement learning-based path optimization and decision-making for penetration testing

3.1 Enhanced Learning Algorithm

3.1.1 DRL algorithm

Deep Reinforcement Learning (DRL) is a subset of Reinforcement Learning that combines deep learning techniques to learn from high dimensional perceptual inputs. In reinforcement learning, an intelligent body interacts with its environment by taking actions and receiving rewards or punishments based on those actions. The goal of the intelligent body is to learn a strategy that maximizes the expected long-term reward. The advantage of DRL over RL is that it handles high dimensional and complex perceptual inputs more efficiently. Conventional reinforcement learning methods might encounter challenges when the state space is extensive or the inputs are intricate.Deep strengthening study (DRL) utilizes the advanced deep studying methods, such as nerve networks, for calculating value functions or making policies. This method causes a more simplified and efficiency-giving way of handling high-dimensional input data.

A further benefit of DRL lies in its capacity to acquire a hierarchical portrayal of inputs. This capability can result in more productive and impactful learning. This is because deep neural networks can learn ways to represent inputs at different levels of abstraction, which allows intelligences to learn more complex and abstract state representations.

Several prevalent Deep Reinforcement Learning (DRL) algorithms encompass Deep Q-Network (DQN), Policy Gradient Techniques, and Actor-Critic Approaches. DQN is a value - centered method. It acquires the optimal action - value function by estimating the Q - function via a deep neural network. The policy gradient method is a policy - based technique that directly refines the policy of an intelligent agent.The Actor-Critic approach is a value- and strategy-based approach that uses two neural networks, one for estimating the value function and one for estimating the strategy.Deep reinforcement learning (DRL) has found application in diverse fields, including robotics, gaming, and natural language processing. It has achieved state-of-the-art performance in several areas, including playing Atari games and controlling robots.

Similarly, DRL is an effective tool for finding the best path for penetration testing as it can handle high dimensional and complex environments with uncertain dynamics. In penetration testing scenarios, attackers need to avoid detecting, navigating the network or exploiting vulnerabilities to achieve an attack on a specific target in a large number of network nodes and extremely complex network connectivity.

3.1.2 DQN algorithm

Deep Q-Network (DQN) is a deep reinforcement learning (DRL) method that takes value as the core. This thing has combined Q-learning, which is one reinforcement learning (RL) method that already got very good establishment, together with deep neural networks. Q-learning carries out work through repeatedly renewing an estimate of the best action-value function. This updating procedure depends upon the rewards which have been observed and the transitional changes between different states. The action-value function, that can be also named as the Q-function, expresses the anticipation of long-term reward that one can get when carrying out a particular action inside a certain given state.

The Deep Q-Network (DQN) employs a complex neural network for the approximation of the function. The condition acts as the input for the neural network, hence the output is a

collection of values for every action, which is presented in the form of a vector. The Q function is learned by minimizing the difference between the predicted Q value and the target Q value calculated using the Bellman equation.

Bellman's equation:

$$V(s) = max_a \left(R(s,a) + \gamma * \Sigma_{s'} T(s,a,s') * V(s') \right) \quad (6)$$

One of the key innovations of DQN is the use of empirical playback, where a large amount of transformation data is stored in a playback buffer and randomly sampled during training. Empirical playback reduces the correlation between successive transformations and allows for more efficient learning.

Another innovation of DQN is the use of a target network, which is a copy of the Q network that is updated less frequently. The target network is used to compute the target Q -value, making training more stable and less prone to fluctuations. It has additionally been expanded to multiple different forms, including Double Deep Q - Network, Dueling Deep Q - Network, and Rainbow Deep Q - Network.

3.1.3 Strategic Gradient Algorithm

The policy gradient method is a reinforcement learning algorithm that directly learns policy functions, which map states to actions. In contrast to value - based approaches like learning action - value functions or state - value functions, policy gradient techniques directly fine - tune the policy function to achieve the maximum anticipated cumulative reward.

One policy function usually is parameterized through a group of parameters that can be learned. The objective of policy gradient approaches is to seek out the best parameters which can make the expected accumulated reward reach its maximum. This thing is obtained through computing the gradient of the expected accumulative reward with regard to the policy parameters, and adjusting the parameters toward the direction of the gradient. The gradient of the expectation cumulative reward with respect to the policy parameters can be obtained estimation by utilization of the policy gradient theorem. The policy gradient theorem tells us that the gradient of the expected accumulated reward with respect to the policy parameters is in proportion to the product of the advantage function and the gradient of the log-probability of the policy. The advantage function is to measure how much better or worse one action is when it compares with the average action in a given certain state. This tool is utilized for considering the influence that the action has upon future rewards. The gradient of the logarithm of the probability which is related to a policy is utilized to encourage the policy that it select actions with higher probabilities, which therefore are likely to lead to greater accumulated rewards.

Strategy gradient methods possess an advantage compared with DQN, because they can directly carry out fine adjustment on the strategy function, so as to enable the maximization of the expected accumulated reward. By way of comparison, the DQN obtains the optimal policy through an indirect way by carrying out approximation of the Q-function. Therefore, the methods of strategy gradient are hence proven to be more efficacious when problems in spaces of high dimension are being handled.

Policy gradient methods are also more suitable for problems with stochastic and non-diminishable environments, where the gradient-based approach of policy gradient methods can provide more robust and reliable learning.

Moreover, policy gradient techniques are capable of learning policies that are more efficient in terms of sample usage, which means that they can learn optimal policies through

less interaction with the environment than DQNs. This is because policy gradient methods can learn from each interaction with the environment, whereas DQNs require many interactions to learn the exact Q-function.

3.2 Penetration Test Path Based on ET-Dueling DQN Algorithm

3.2.1 ET-Dueling DQN Algorithm Modeling

In this section, a Dueling DQN algorithm called Experience Campaigning (DQN) is proposed, which is an improved version of the traditional Dueling DQN algorithm. The traditional Dueling DQN algorithm is defective in terms of experience playback pool selection strategy, while the DQN algorithm with experience campaigning optimizes the selection strategy of the traditional algorithm by selecting the experience playback pool through the experience campaigning mechanism.

Dueling DQN algorithm is an algorithm in the field of deep reinforcement learning for solving decision-making problems in discrete state and action spaces, which is commonly used for problems in similar fields such as game intelligence. The fundamental concept of the algorithm involves decomposing the value function (Q - function) into a state value function (V - function) and an advantage function (A - function). This decomposition enables the algorithm to separately assess the value of each state and action. In Dueling DQN, two sub - network architectures are presented, which respectively correspond to the aforementioned value function network and advantage function network, as depicted in Fig. 2. The output of the final Q - network is derived through a linear combination of the outputs of the value function network and the dominance function network. The key benefit of this algorithm lies in its ability to more effectively manage intricate state and action spaces, resulting in superior performance and outcomes.

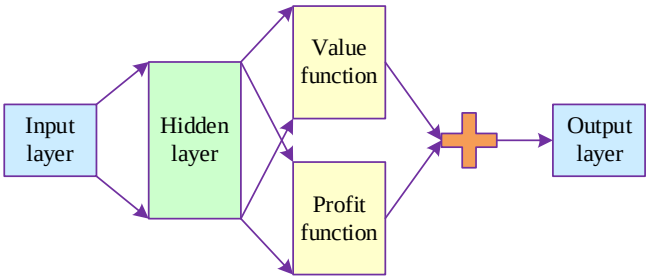


Figure 2: Dueling DQN network structure diagram

Specifically, for a given state, the Dueling DQN algorithm outputs the state value function and the dominance function of that state through two neural networks, respectively, and then combines the two functions to obtain the final Q value. The first part is related only to the state s and not to the specific action a to be adopted, which is called the value function part and is notated as $V(s,w,\alpha)$, and the second part is related to both the state s and the action a, which is called the dominance function part and is notated as $A(s,w,\alpha,\beta)$, then the value function of the final Dueling DQN algorithm can be re-expressed as Equation (7):

$$Q(s,a,w,\alpha,\beta)=V(S,w,\alpha) + \left(A(s,a,w,\beta)-\frac{1}{\mathcal{A}}\sum_{a'\in\mathcal{A}}A(s,a',w,\beta)\right) \tag{7}$$

In this way, the algorithm can better distinguish the value of each action and learn the advantages of each action faster.The learning procedure of the Dueling DQN algorithm bears resemblance to that of the conventional DQN algorithm, which improves the training efficiency and stability of the algorithm by using the experience playback and the target network. Meanwhile, in each training step, the Dueling DQN algorithm selects the action with the largest Q value and explores other actions with the $\epsilon-greedy$ strategy to ensure the convergence and exploration of the algorithm.

In Dueling DQN, TD (Temporal Difference) learning is a difference-based learning method that utilizes the difference between the current and future estimates to adjust the estimates.TD-error is the difference between the predicted value of an action and the actual reward obtained in the current state, whereby the TD-error: δ can be expressed as in Eq. (8). Where θ and θ^{-} represent the parameter values of the target and the target network respectively, r represents the reward value, and γ represents the decay factor:

$$\delta = r + \gamma \max_{a} Q\left(s, a, \theta^{-}\right) - Q\left(s, a, \theta\right) \tag{8}$$

Experiences with high immediate return values have a significant impact on agent learning, so this section uses TD-error and immediate return values as a measure of sample prioritization.The larger the TD-error immediate return value, the higher the priority. The TD-error and immediate return value r are prioritized as y_1 and y_2, respectively, with y_1 shown in Equation (9) and y_2 shown in Equation (10):

$$y_1 = \frac{|\delta| - |\delta|^{\min}}{|\delta|^{\max} - |\delta|^{\min}} \tag{9}$$

$$y_2 = \frac{r - r^{\min}}{r^{\max} - r^{\min}} \tag{10}$$

where $|\delta|^{\max}$, $|\delta|^{\min}$ represent the maximum and minimum values of the TD-error in the empirical pool and $r^{\max}$, $r^{\min}$ represent the r maximum and minimum values.

The final priority Y is denoted as: $Y = y_1 + y_2$.

3.2.2 Algorithm flow

The flowchart of the campaign strategy is shown in Figure 3. Experience campaigning is a strategy used in reinforcement learning to decide which experiences will be added to the replay buffer. Experience campaigning selects experiences based on certain rules, usually choosing those with high estimates to reduce the risk of overfitting. In the process of Dueling DQN algorithm experience replay, the selection of experience data is random and may not be able to fully utilize all the experience data. The experience campaign mechanism can be filtered according to the value of the experience data and select the most valuable experience data for training, avoiding the repeated training of certain experiences in the experience pool, thus accelerating the convergence speed and improving the efficiency of the algorithm.

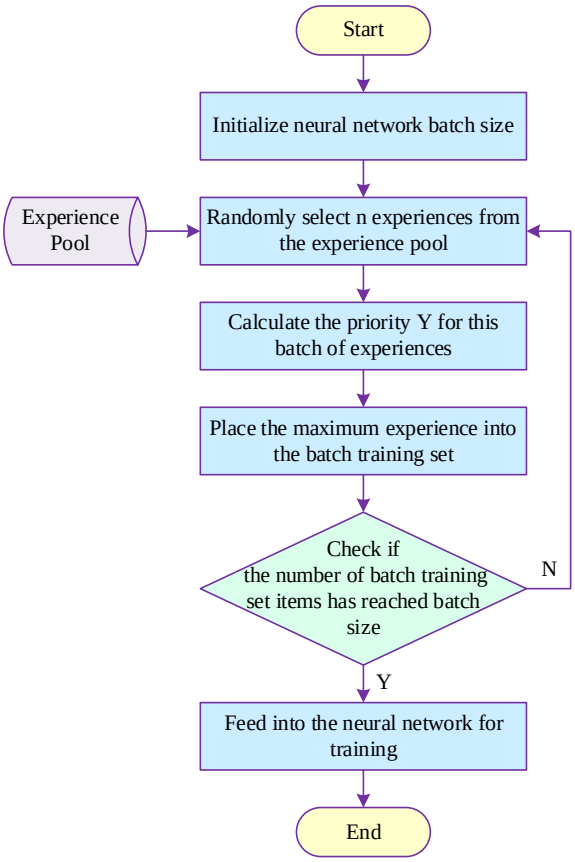


Figure 3: Campaign strategy flowchart. Campaign strategy process

In the campaign strategy process, the batchsize of the neural network is initialized and randomly obtains experiences from the experience pool, calculates the Y-value of each experience, and selects the one with the largest Y-value experience to be put into the batch training set until the batchsize is full, and puts it into the neural network for training. The process flow of the ET - Dueling DQN algorithm is presented as follows:

Inputs: maximum number of steps T, state s, action a, discount factor γ, number of iterations K, experience playback pool D.

Output: Q network parameters.

1: Initialize action-value network Q and state-value network V, target action-value network Q' and target value network V'.

2: Initialize the experience replay pool D.

3: for $episode = 1$ to K do.

4: Initialize s to be the first state of the current state sequence, get the feature vector $\phi(s)$.

5: for $t = 1$ to T do.

6: Choose action a_t with probability ϵ, otherwise choose action randomly.

7: Execute action a_t, observe reward r_t and new state s_{t+1}.

8: Calculate the priority Y of the experience by transferring (s_t, a_t, r_t, s_{t+1}) experience samples and put it into the experience playback pool D.

9: Select n experiences (s_i, a_i, r_i, s_{i+1}) according to the flow of Fig. 3 and put them into

the batch training set to calculate the Q value and V value:

$$\begin{cases} Q_{i,a} = Q\left(s_i, a_i; \theta\right) \\ V_i = V\left(s_i; \theta\right) \end{cases} \tag{11}$$

10: Select n experiences $\left(s_i, a_i, r_i, s_{i+1}\right)$ according to the flow of Fig. 3, and compute the Q' values and V' values:

$$\begin{cases} Q'_{i,a} = Q\left(s_i, a_i; \theta^-\right) \\ V'_i = V\left(s_i; \theta^-\right) \end{cases} \tag{12}$$

11: Compute the dominance function A for each sample:

$$A_i = r_i + \gamma * Q'_{i,argmax_a\left(Q_{i,a}\right)} - V_i \tag{13}$$

12: Calculate the target value:

$$y_i = V'_i + A_i \tag{14}$$

13: Training an action value network using the MSE loss function:

$$L\left(\theta\right) = \frac{1}{N}\sum_i \left[\left(y_i - Q_{i,a_i}\right)^2\right] \tag{15}$$

14: Training an action value network using the MSE loss function:

$$L\left(\theta\right) = \frac{1}{N}\sum_i \left[\left(y_i - V_i\right)^2\right] \tag{16}$$

15: Update the target network parameters:

$$\theta^- = \tau * \theta + \left(1 - \tau\right) * \theta^- \tag{17}$$

16: Update state s to new state s_{t+1}.

17: end for

18: end for

3.3 Penetration Testing Decision Algorithm Based on Q-Function

Penetration testing learning algorithms based on Q-Learning and Dueling DQNs are able to learn action strategies from penetration testing interactions. In order to utilize the learned strategies to guide the penetration test, a penetration test decision-making algorithm based on the Q function is constructed in this section. Based on the analysis of the characteristics of the penetration test process, the algorithm incorporates three sets O, C, and U in addition to utilizing the Q function for selecting the optimal action in the current state of the

penetration test environment. The set O is used to save the detected hosts, the set C is used to save the hosts that have successfully penetrated and established a backdoor, and the set U is used to save the hosts that have not established a backdoor. When faced with multiple hosts, the decision algorithm first adds these hosts to the set O and then infiltrates the hosts in the set O in turn. If the host is successfully controlled, it is removed from set O and saved in set C, otherwise it is stored in set U. In addition, new hosts discovered through scanning probes after controlling the host are stored in set O. These three collections work together to satisfy features 1 and 3, and feature 2 is achieved by removing each action from the action space after it has been performed on a single host during the penetration process.

4 Experiments and analysis of automated penetration test path optimization and decision making

4.1 Algorithmic Penetration Testing Usability Analysis

4.1.1 Penetration success rate

For verifying the penetration success percentage of the arithmetic method this research uses, one aggressor carries out an attack toward the target in the experiment environment. The time interval for network nodes in the target to exchange penetration information is set to 1 minute and the host updates the penetration information. Penetration attack on network nodes in the target using this study, KaliLinux and SSM are tested as comparison experiments, the penetration test time is set to 10 minutes, and the penetration success rate of each algorithm is obtained as shown in Figure 4.

In the process of penetration attack on the target network, the penetration path between the network nodes has diversity, the direct penetration of two network nodes is not necessarily the optimal penetration path, resulting in inconsistent network node penetration success rate. This study penetration test to launch penetration attacks with a high success rate, able to break the network nodes of the target algorithm, and use the occupied nodes to continue to launch attacks on other network nodes, penetration of the target algorithm of the various nodes of the success rate is higher than 80%, of which the No. 4 network node penetration success rate of the highest reached 95.415%. Literature KaliLinux launched penetration attacks on each network node, the penetration success rate is lower, including the lowest penetration success rate of network node No. 3 is 68.748%, and the highest penetration success rate on network node No. 6 reaches 81.154%, and the penetration success rate of SSM is generally higher than 70%, with network node No. 4 reaching the highest penetration success rate of 85.124%, which is still in line with the present research There is a large difference in the success rate of

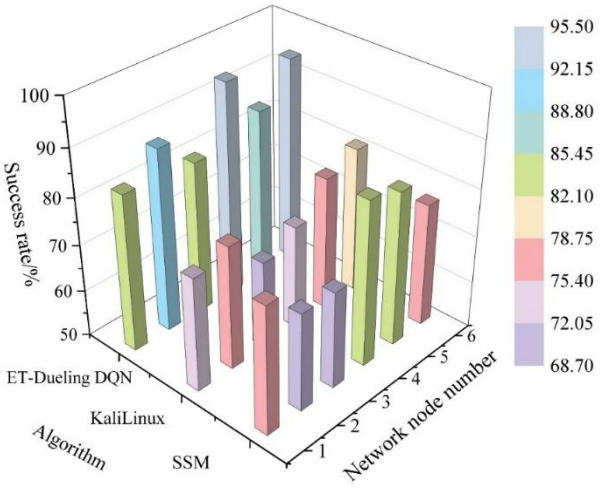


Figure 4: Penetration rate

4.1.2 Value of the attack

Due to the different attack nodes and different distances from the attack target, it leads to some differences in the attack value of performing penetration attacks. Using this study to generate the global attack graph, infiltration attack on the attack target, so that the calculation of the attack value, the three algorithms on different target hosts of the attack value is shown in Figure 5.

Due to the small size of the generated global attack graph, the attack paths may be the same for the same target hosts, and the obtained attack values are not much different. In this study, the attack value of the attack path generated based on the structure of the target network to launch the penetration attack is higher, up to 27.348, and the value of the attack on the target host No. 4 is lower at 23.422. The attack value of KaliLinux for launching an infiltration attack on target host #6 is minimized to 20.315, which is low compared to other algorithms, and the attack path of KaliLinux is not the optimal attack path. The attack value of SSM on target hosts #4, #5, and #6 is close to that of this study, with a maximum of 24.965, but target hosts #1 and #2 have the lowest attack value, and the attacking node's pre-attribute nodes have redundant information.

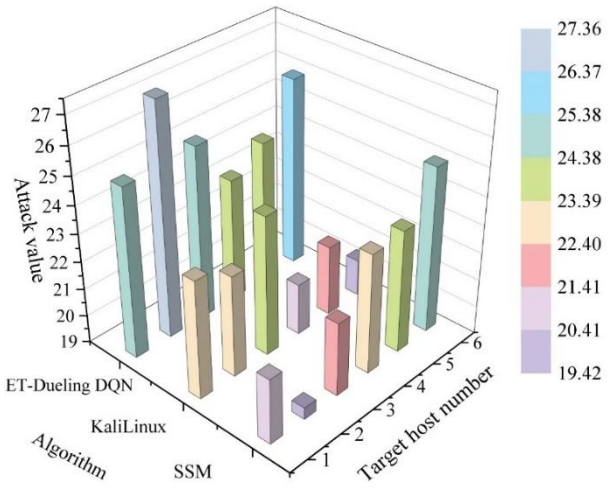


Figure 5: Attack value

4.2 Penetration Test Performance Simulation

4.2.1 Experiment and Parameter Settings

The attacker can only access subnet 1 at the initial moment, regarding whether there is a network connectivity relationship between the subnets can be realized by configuring firewall rules. Set subnet 1 can access subnet 3 cannot access subnet 2, subnet 2 can only access subnet 3 cannot access subnet 1, and subnet 3 can access subnet 1 as well as subnet 2. The attacker is not aware of the existence of subnets 2 and 3 at the initial moment, and the corresponding state value is 0 in the state space, and when subnet 3 is discovered in subnet 1 through the information-gathering action, the corresponding state value is changed to 1. Based on this Vulnerability exploitation can be performed on the hosts of subnet 3 to modify the corresponding state value, and so on. Three network environments of different sizes are first designed to test the performance of this paper's algorithm against other algorithms. The number of subnets, the number of subnets per host, and the number of host vulnerabilities in each network environment gradually increases, while the complexity of the network structure increases, and the attacker needs more attack steps to reach the target state. The target hosts are selected as 2 from the last two subnets, and the asset value is set to 10 for both.

The experimental hardware configuration includes Intel i9-7980XE CPU, 128G RAM, the operating system is windows 10, and the algorithm program is written through python language. Some hyperparameters need to be adjusted appropriately according to the expansion of the problem scale, and the setting of such hyperparameters has a large impact on the experimental results.

4.2.2 Presentation and analysis of experimental results

Motivated by the reward worth, the intelligent entity acquires the strategy for the highest cumulative reward value via continuous experimentation and learning from mistakes.Initially, without learning, the intelligent body tends to use a random strategy to select actions to execute for unknown environments, uses a larger average number of training steps per round, obtains a smaller cumulative reward value, and intrudes into honeypot hosts, but as it continues to explore and learn, behaviors that result in a decrease in the cumulative reward value (e.g., excessive actions or intruding into honeypot hosts) are not learned, and ultimately, the intelligent body learns to attack sensitive hosts with a positive reward value using the least number of actions to attack sensitive hosts with positive reward values.

In this paper, in order to test the scalability of the algorithm under different sizes of networks, the number of hosts in the network is increased based on scenario one and the rest of the network scenarios are constructed.

The same set of hyperparameters are used to test the four algorithms DDQN, H_DDQN, DH_DDQN, and Dueling_DDQN under the experimental scenario I. DQN is used as the benchmark to compare and measure the change in the average reward value of the algorithms over a specified number of training steps, the probability of the honeypot hosts to be compromised, and the change in the number of training steps used in each round.

Fig. 6 shows the variation of the average cumulative reward value with the number of training steps. The Dueling_DDQN algorithm constructed in this paper reaches the task of 200 average cumulative reward value before the number of training steps is 0.4×10^6, while DH_DDQN, H_DDQN, and DDQN reach it around 0.405×10^6, 0.41×10^6, and 0.46×10^6, respectively. The advantage of the average cumulative reward value of the algorithm designed in this paper is shown.

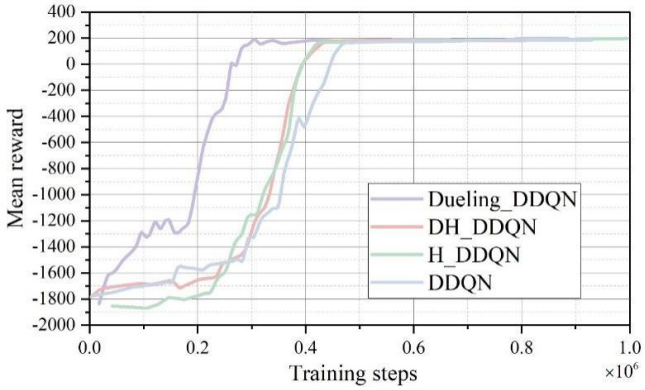


Figure 6: Mean reward versus training steps

Figure 7 shows the change of the invasion probability of the honeypot host as regarding the quantity of training steps. Along with the increase of the training step number,the probability of honeypot host intrusion using the Dueling_DDQN algorithm decreases gradually, and completes the process of changing from 1→0 probability before the number of training steps reaches 0.3×10^6.

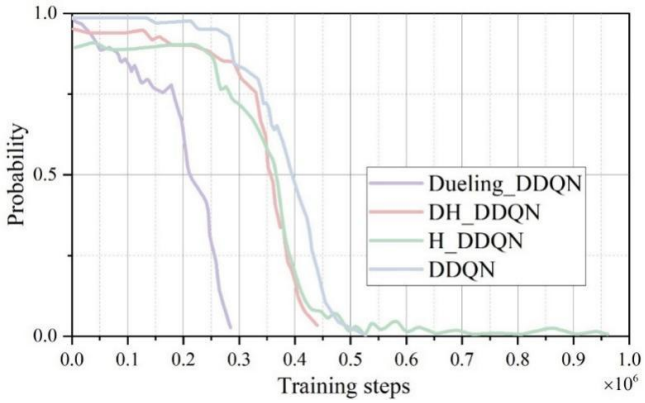


Figure 7: The receiver is affected by the probability of the invasion

Figure 8 shows the variation of the number of steps used in each round of training, the Dueling_DDQN algorithm uses only 0.345×10^6 training steps from 2000 to 0, which is the algorithm that uses the fewest number of training steps among the four methods and achieves the training effect the fastest.

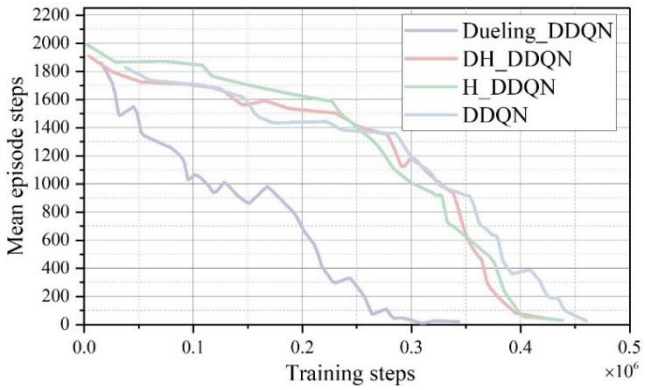
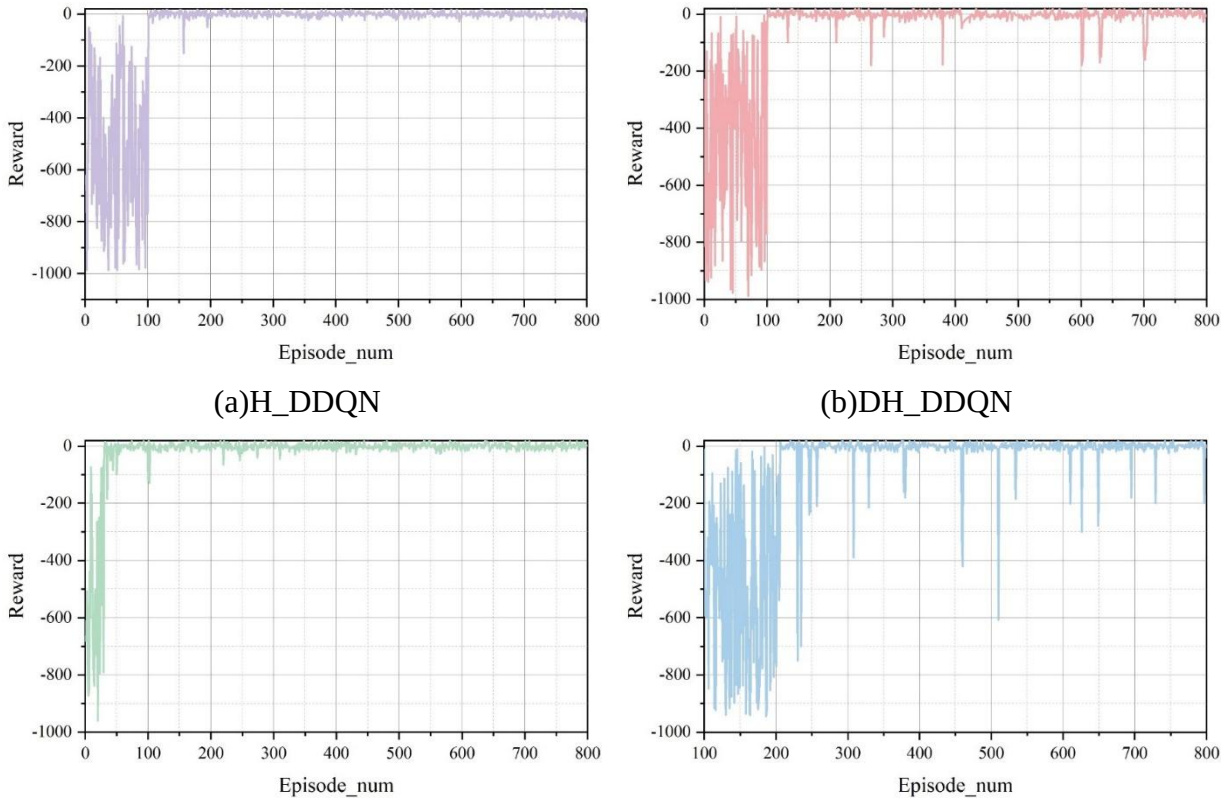


Figure 8: Changes in the number of steps per leg training

4.2.3 Optimal environmental reward values for different scenarios

The experimental results from Scenario 1 show that the four algorithms have a similar number of initial convergence in the case of smaller experimental scale, while the Dueling_DDQN algorithm has a more rapid learning process.

Fig. 9 shows the experimental results of scenario 2, with H_DDQN in Fig. (a), DH_DDQN in Fig. (b), Dueling_DDQN in Fig. (c), and DDQN in Fig. (d).The network size is scaled up, and the learning process is more complex and lengthy for the DH_DDQN and H_DDQN algorithms, and the DDQN takes even longer than the two algorithms.The Dueling_DDQN algorithm is capable of learning in a much more rapid manner than the DDQN algorithm. DDQN algorithm is able to reach convergence in fewer number of trainings and reaches the desired reward value after 50 trainings.



(a)H_DDQN (b)DH_DDQN

(c)Dueling_DDQN (d)DDQN

Figure 9: Scene 2 results

Figure 10 shows the running time of the training process of the four algorithms; the DDQDN algorithm grows too fast and converges slowly in the test environment with a large state space, and the Dueling_DDQN algorithm has the fastest running time, with the time needed to reach convergence at around 22s.

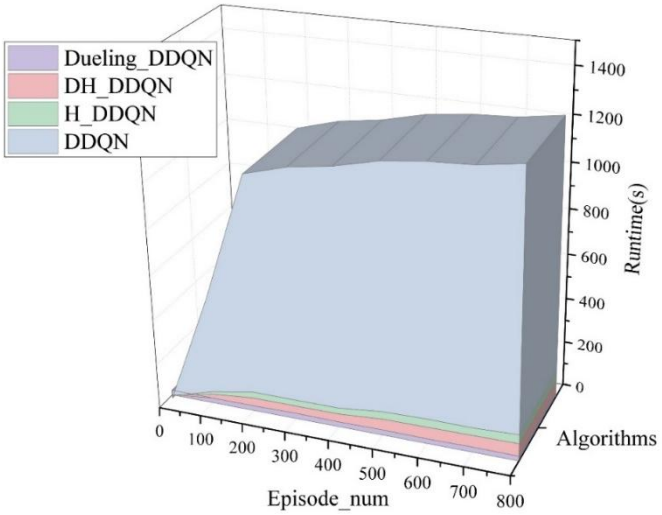


Figure 10: Four algorithm training process running time

The number of iterations and the running time when the algorithm approximates convergence are shown in Fig. 11. Scenarios 3, 4, and 5 increase the total number of host vulnerabilities while keeping the total number of network topologies consistent and use the algorithm of this paper for path planning. Scenarios 3, 6, and 7 increase the number of subnets and hosts while keeping the total number of host vulnerabilities and use the algorithm of this paper for path planning. With the number of networks and hosts unchanged, as the number of host vulnerabilities increases, the algorithm requires more iterations to reach approximate convergence, and the time growth is relatively smooth, with the number of iterations required increasing from 132 to 284, and the time required stabilizing in the range of about 100~130s. As the quantity of subnets and the number of hosts within each subnet increase, the time necessary for the algorithm to reach convergence surges from 100.26 seconds to 3314.45 seconds. This is due to the fact that the number of actions the Agent has to attempt in each plot escalates rapidly.In a network environment with less than 100 hosts, the algorithm in this paper is able to find a planning solution in a finite range of time, and it is able to realize the automated penetration test path for the optimization and decision making.

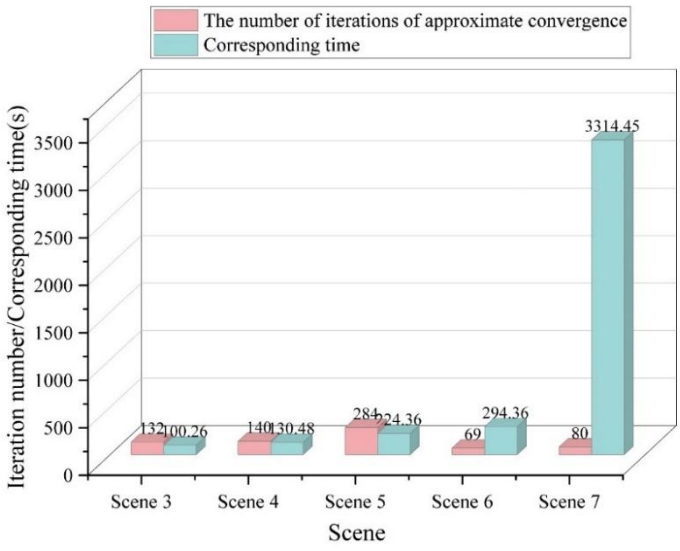


Figure 11: The number of iterations and the running time of the algorithm approximation

5 Conclusion

This study is based on a Markov decision process definition that takes the maximum cumulative reward as a reward, from which an action strategy is determined and an optimal strategy is learned to serve the maximum expected reward. Considering the process of penetration against a single host, the state, action, and reward functions are defined separately to enable the modeling of penetration testing. Based on deep reinforcement learning algorithm, an empirical campaigning Dueling DQN algorithm is proposed, combined with Q-Learning algorithm, to establish a penetration test decision-making algorithm based on Q-function, to complete the automated penetration test path optimization and decision-making.

The feasibility of the algorithm penetration test for experimental analysis, this study penetration test to launch penetration attacks with a high success rate, for the target algorithm of the nodes penetration success rate is higher than 80%, of which the penetration success rate of the No. 4 network node is 95.415%. The highest penetration success rates of the remaining mainstream algorithms are 81.154% and 85.124%, respectively, which are still quite different from the success rate of this study.

The Dueling_DDQN algorithm constructed in this study achieves the task with an average cumulative reward value of 200 before the number of training steps is 0.4×10^6, which is significantly better than the DH_DDQN, H_DDQN, and DDQN algorithms, and thus the advantages of the algorithm designed in this paper are reflected in the average cumulative reward value.

About the Author

Wei Li was born in Dezhou, Shandong, P.R. China, in 1982. He is a senior engineer at the Digital Intelligence Technology Company, PetroChina Xinjiang Oilfield Company. His main research direction is cybersecurity.

Zixuan Zhao was born in Karamay, Xinjiang, P.R. China, in 1995. He is an engineer at the Digital Intelligence Technology Company, PetroChina Xinjiang Oilfield Company. His main research direction is cybersecurity.

Youchen Shi was born in Karamay, Xinjiang, P.R. China, in 1990. He is an engineer at the Digital Intelligence Technology Company, PetroChina Xinjiang Oilfield Company. His main research direction is cybersecurity.

Yinquan Wang was born in Karamay, Xinjiang, P.R. China, in 1988. He is an Engineer at the Digital Intelligence Technology Company, PetroChina Xinjiang Oilfield Company. His main research direction is cybersecurity.

Zeyang Zhao was born in Weinan, Shanxi, P.R. China, in 1992. He is an engineer at the Digital Intelligence Technology Company, PetroChina Xinjiang Oilfield Company. His main research direction is cybersecurity.

Hanlin Tu was born in Bazhong, Sichuan, P.R. China, in 2000. He is an assistant engineer at the Digital Intelligence Technology Company, PetroChina Xinjiang Oilfield Company. His main research direction is cybersecurity.

References

[1] Pereira, T., Barreto, L., & Amaral, A. (2017). Network and information security challenges within Industry 4.0 paradigm. Procedia manufacturing, 13, 1253-1260.

[2] Hawamleh, A. M. A., Alorfi, A. S. M., Al-Gasawneh, J. A., & Al-Rawashdeh, G. (2020). Cyber security and ethical hacking: The importance of protecting user data. Solid State Technology, 63(5), 7894-7899.

[3] Esteves, J., Ramalho, E., & De Haro, G. (2017). To improve cybersecurity, think like a hacker. MIT Sloan Management Review.

[4] Humayun, M., Niazi, M., Jhanjhi, N. Z., Alshayeb, M., & Mahmood, S. (2020). Cyber security threats and vulnerabilities: a systematic mapping study. Arabian Journal for Science and Engineering, 45(4), 3171-3189.

[5] Akram, J., & Ping, L. (2020). How to build a vulnerability benchmark to overcome cyber security attacks. IET Information Security, 14(1), 60-71.

[6] Mhara, M. A. A., Abdulrahman, A. A., & Baroud, A. A. (2024). Cyber attacks and threats: Study of the types of cyber attacks: Hacking, viruses, targeted attacks, and electronic espionage. Int. J. Electr. Eng. and Sustain., 38-47.

[7] Amer, L. (2025). AI in cyber security: A dual perspective on hacker tactics and defensive strategies. Cyber Security: A Peer-Reviewed Journal, 8(3), 198-213.

[8] Ko, R. K. (2020). Cyber autonomy: automating the hacker–self-healing, self-adaptive, automatic cyber defense systems and their impact on industry, society, and national security. In Emerging technologies and international security (pp. 173-191). Routledge.

[9] Wu, Y., Xiao, H., Dai, T., & Cheng, D. (2022). A game-theoretical model of firm security reactions responding to a strategic hacker in a competitive industry. Journal of the Operational Research Society, 73(4), 716-740.

[10] Sarker, K. U., Yunus, F., & Deraman, A. (2023). Penetration taxonomy: A systematic review on the penetration process, framework, standards, tools, and scoring methods. Sustainability, 15(13), 10471.

[11] Filiol, E., Mercaldo, F., & Santone, A. (2021). A method for automatic penetration testing and mitigation: A red hat approach. Procedia Computer Science, 192, 2039-2046.

[12] Song, B., Sun, L., & Qin, Z. (2022). Design of web security penetration test system based on attack and defense game. Scientific Programming, 2022(1), 8645969.

[13] Li, Y., & Li, X. (2021). Research on Multi-Target Network Security Assessment with Attack Graph Expert System Model. Scientific Programming, 2021(1), 9921731.

[14] Zhong, C., Yen, J., Liu, P., & Erbacher, R. F. (2018). Learning from experts' experience: toward automated cyber security data triage. IEEE Systems Journal, 13(1), 603-614.

[15] Diana, L., Dini, P., & Paolini, D. (2025). Overview on intrusion detection systems for computers networking security. Computers, 14(3), 87.

[16] Altulaihan, E. A., Alismail, A., & Frikha, M. (2023). A survey on web application penetration testing. Electronics, 12(5), 1229.

[17] Krishnama, S. (2023). A process of penetration testing using various tools. Mesopotamian Journal of CyberSecurity, 2023, 93-103.

[18] Armstrong, M. E., Jones, K. S., Namin, A. S., & Newton, D. C. (2018, September). The knowledge, skills, and abilities used by penetration testers: Results of interviews with cybersecurity professionals in vulnerability assessment and management. In Proceedings of the Human Factors and Ergonomics Society Annual Meeting (Vol. 62, No. 1, pp. 709-713). Sage CA: Los Angeles, CA: SAGE Publications.

[19] Kaur, G., & Kaur, N. (2017). Penetration testing–reconnaissance with NMAP tool. International Journal of Advanced Research in Computer Science, 8(3), 844-846.

[20] Ren, Q., Liu, J., Xiong, X., & Lu, C. (2024, July). Automated Penetration Testing Based on LSTM and Advanced Curiosity Exploration. In 2024 IEEE 5th International Conference on Pattern Recognition and Machine Learning (PRML) (pp. 150-156). IEEE.

[21] Ren, Q., Xiong, X., Liu, J., & Wang, Z. (2024, November). Automated Penetration Testing Based on MATD Algorithm. In 2024 4th International Conference on Artificial Intelligence, Virtual Reality and Visualization (pp. 150-155). IEEE.

[22] Samad, A., Altaf, S., & Arshad, M. J. (2024). Advancements in Automated Penetration Testing for IoT Security by Leveraging Reinforcement Learning. evaluation, 8, 9.

[23] Li, Q., Wang, R., Li, D., Shi, F., Zhang, M., Chattopadhyay, A., ... & Li, Y. (2024). Dynpen: Automated penetration testing in dynamic network scenarios using deep reinforcement learning. IEEE Transactions on Information Forensics and Security, I 8966-8981.

[24] Moreno, A. C., Hernandez-Suarez, A., Sanchez-Perez, G., Toscano-Medina, L. K., Perez-Meana, H., Portillo-Portillo, J., ... & García Villalba, L. J. (2025). Analysis of autonomous penetration testing through reinforcement learning and recommender systems. Sensors, 25(1), 211.

[25] Sun, S., Lu, Y., Wu, D., & Zhang, G. (2025). Intelligent penetration testing method for power internet of things systems combining ontology knowledge and reinforcement learning. PLoS One, 20(5), e0323357.

[26] Shen, X., Wang, L., Li, Z., Chen, Y., Zhao, W., Sun, D., ... & Ruan, W. (2025, August). Pentestagent: Incorporating llm agents to automated penetration testing. In Proceedings of the 20th ACM Asia Conference on Computer and Communications Security (pp. 375-391).

[27] Deng, G., Liu, Y., Mayoral-Vilches, V., Liu, P., Li, Y., Xu, Y., ... & Rass, S. (2024). {PentestGPT}: Evaluating and harnessing large language models for automated penetration testing. In 33rd USENIX Security Symposium (USENIX Security 24) (pp. 847-864).

[28] Wang, W., Gu, H., Wu, Z., Chen, H., Chen, X., & Shi, F. (2025). PTFusion: LLM-driven Context-Aware Knowledge Fusion for Web Penetration Testing. Information Fusion, 103731.

[29] Alghamdi, F. (2025). Automated Penetration Testing Through Reinforcement Learning. In Complexities and Challenges for Securing Digital Assets and Infrastructure (pp. 323-352). IGI Global Scientific Publishing.