Boosting Reinforcement Learning via Hierarchical Game Playing With State Relay

Chanjuan Liu, *Member, IEEE*, Jinmiao Cong, Guangyuan Liu, Guifei Jiang, Xirong Xu, and Enqiang Zhu

***Abstract*— Due to its wide application, deep reinforcement learning (DRL) has been extensively studied in the motion planning community in recent years. However, in the current DRL research, regardless of task completion, the state information of the agent will be reset afterward. This leads to a low sample utilization rate and hinders further explorations of the environment. Moreover, in the initial training stage, the agent has a weak learning ability in general, which affects the training efficiency in complex tasks. In this study, a new hierarchical reinforcement learning (HRL) framework dubbed hierarchical learning based on game playing with state relay (HGR) is proposed. In particular, we introduce an auxiliary penalty to regulate task difficulty, and one training mechanism, the state relay mechanism, is designed. The relay mechanism can make full use of the intermediate states of the agent and expand the environment exploration of low-level policy. Our algorithm can improve the sample utilization rate, reduce the sparse reward problem, and thereby enhance the training performance in complex environments. Simulation tests are carried out on two public experiment platforms, i.e., MazeBase and MuJoCo, to verify the effectiveness of the proposed method. The results show that HGR significantly benefits the reinforcement learning (RL) area.**

***Index Terms*— Curse of dimensionality, game theory, hierarchical reinforcement learning (HRL), sampling efficiency.**

I. Introduction

A. Background

THE development of deep reinforcement learning (DRL) [1] has brought about a new era of artificial intelligence. Through DRL, agents can handle a variety of problems faster and better than humans, such as motion planning [2], obstacle avoidance [3], [4], game playing [5], and autonomous driving [6].

Manuscript received 10 October 2022; revised 13 September 2023 and 13 January 2024; accepted 30 March 2024. This work was supported in part by the National Key Research and Program of China under Grant 2021ZD0112400; and in part by the National Natural Science Foundation of China under Grant 62172072, Grant 62272115, and Grant U1908214. *(Corresponding author: Enqiang Zhu.)*

Chanjuan Liu is with the Cancer Hospital of Dalian University of Technology, School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China (e-mail: chanjuanliu@dlut.edu.cn).

Jinmiao Cong, Guangyuan Liu, and Xirong Xu are with the School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China (e-mail: cjm111@mail.dlut.edu.cn; lgy_graduate@163.com; xirongxu@dlut.edu.cn).

Guifei Jiang is with the College of Software, Nankai University, Tianjin 300350, China (e-mail: G.Jiang@nankai.edu.cn).

Enqiang Zhu is with the Institute of Computing Science and Technology, Guangzhou University, Guangzhou 510006, China (e-mail: zhuenqiang@gzhu.edu.cn).

Digital Object Identifier 10.1109/TNNLS.2024.3386717

One advantage of DRL for agents is that the agents do not obsess over the state of the environment itself but rather adjust their policies according to the reward feedback of the environment. Then, the goal is achieved by learning the policies. Therefore, an important issue of reinforcement learning (RL) is to design an appropriate reward function. In actual applications, it is difficult for the agents to obtain enough effective rewards, or the rewards are sparse, leading to slow and even ineffective learning. Coupled with the large combination of state and action space, the curse of dimensionality has always been a problem in DRL.

B. Related Work and the Challenge

Since the 1990s, hierarchical reinforcement learning (HRL) [7], [8], [9] has gained increasing attention, driven by the advancements in deep learning and neural networks. HRL solves complex tasks using hierarchical decompositions, providing a feasible solution for the curse of dimensionality and sparse rewards. HRL involves high-level policies that learn to select appropriate subtargets, and then low-level policies implement corresponding actions [10]. In general, HRL can be classified into two main categories based on training methods: hierarchical training and end-to-end training.

Hierarchical training HRL trains the model in a hierarchical manner, which decomposes the learning tasks into multiple levels, with each level responsible for addressing different abstract levels of the task. Kulkarni et al. [11] proposed a two-layer hierarchical deep Q-learning (H-DQN) model that can solve the issue of sparse rewards in deep Q-networks (DQNs) [12]. On the other hand, Vezhnevets et al. [13] applied feUdal networks to HRL, which consists of a manager module and a worker module, to address the long-standing problems of credit distribution and sparse rewards. Both these methods are particularly effective for playing the Montezuma's revenge game on Atari. To improve its generalizability across similar tasks, Florensa et al. [14] proposed a skill-based stochastic neural network, namely, stochastic neural networks for HRL (SNN4HRL). This model combines two training processes: an unsupervised process for learning many skills using proxy rewards and a small amount of expert domain knowledge and a high-level training process that encapsulates the learned skills for reuse in future tasks.

There are also some studies on end-to-end training HRL. In [15], [16], and [17], hindsight action and goal transitions were used to enable the agent to learn multiple policies simultaneously. This approach addressed the problems of sparse rewards and instability in parallel training of multilayer policies. In [18] and [19], subgoal discovery was used to find appropriate subgoals. However, they often encountered training inefficiency as the goal space was vast. Discovering useful goals in a large space poses challenges for both high-level subgoal generation and low-level policy learning. To alleviate the problem of a vast goal space, Zhang et al. [20] proposed restricting the high-level action space from the whole goal space to a k-step adjacent region of the current state using an adjacency constraint. By doing so, the agent can gradually learn suitable policies based on effective subgoals proposed by the high level.

Each of the current HRL algorithms has its advantages, yet most algorithms are limited to specific environments and have poor generalization capabilities. To this end, game theory [21], [22], [23] has been continuously applied to RL algorithms to improve their performance in different scenarios [24]. One example is self-play, which has achieved excellent performances in checkers [25], TD-gammon [26], and Go [27], [28]. To address the issue of low sample efficiency in model-free RL, Sukhbaatar et al. [29] carried out unsupervised agent training with intrinsic motivation and automatic curriculum learning via self-play, which enabled the agent to explore the environment autonomously and make continuous improvements. Sukhbaatar et al. incorporated the unsupervised method of [29] into HRL and proposed an asymmetrical self-play learning method, called hierarchical self-play (HSP) [30], to improve the algorithm's performance.

However, in the current game-based HRL algorithms, when an agent works on a task during the game, regardless of task completion, the environment state in which the agent is situated will be reset afterward. As a result, much of the state information is lost, leading to a low sample utilization rate and hindering further explorations of the environment. Moreover, in the initial pretraining stage, the agent generally has a weak learning ability and can only complete simple tasks. As a result, if the agent responsible for task allocation assigns a difficult task, it may adversely affect the training efficiency and effectiveness.

C. Our Contribution

To solve the aforementioned problems in game-based HRL, an HRL algorithm based on game playing with the mechanism of state relay (HGR) is proposed in this work.

Our algorithm comprises two distinct stages of training: low-level training and high-level training. The low-level training is a pretraining stage. During the pretraining stage, the policies are trained to explore and familiarize the environment through games. The second training is a fine-tuning stage, in which the high-level policies aim to propose subgoals for the low-level policies. We use a backpropagation (BP) [31] network to model the high-level policies and error BP is used during the training process to correct the model and speed up loss function convergence.

In the pretraining stage, low-level policies are unfamiliar to the environment before the training of high-level policies. The agents need to understand the environment through a long period of trial and error. This leads to a low sample utilization rate and a long training process. To overcome this, two game-playing agents, A and B, are introduced in low-level policy training. Agent A is responsible for proposing tasks, and agent B completes them. Rewards are given to either agent A or B. Through this competitive reward structure, agent A will continuously propose new tasks that have not been explored to ensure task diversity. Then, a maximum step length is set for the tasks such that it covers all the tasks of a given difficulty in the entire environment. However, if agent A continues proposing difficult tasks that agent B cannot complete, the training will be meaningless. To prevent this, we introduce an auxiliary penalty: agent A will be penalized in the above-mentioned case (e.g., if A proposes four tasks that B cannot complete continuously, it will be punished -2 or more. Details can be found in Section III-A). Through receiving this penalty feedback, A will be able to adjust the difficulty of the tasks it proposes actively.

To better explore the environment, a state relay mechanism is adopted. If agent B successfully completes the training task, the current state will temporarily be retained rather than reset. By doing so, the agent can continue exploring new tasks. Thus, the scope of environmental exploration is expanded, and the agent's ability to complete tasks is enhanced.

Our method was tested on two complex environments, including AntGather, to illustrate its effectiveness. The results indicate that our proposed algorithm can bolster the performance of the existing motion planning HRL algorithms.

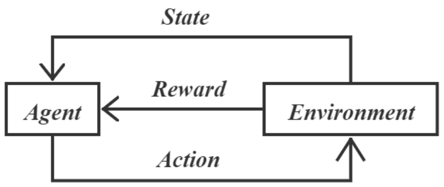


Fig. 1. RL process.

D. Organization of This Article

The rest of this article is organized as follows. Section II presents the preliminaries of RL, followed by an introduction of the proposed method in Section III. Section IV describes the experimental environment and comparatively analyzes the experimental results. Finally, Section V concludes this article and presents future research directions.

II. Preliminaries

A. Reinforcement Learning

As shown in Fig. 1, RL [32], [33], [34] refers to the continuous interaction between the agent and the environment.

Through continuous trial and error, the agent adjusts its next action based on the reward for each interaction. The goal of RL is to learn a reward maximization policy π for a given Markov decision process (MDP). Generally, a five-tuple $\langle S, A, P, R, \gamma \rangle$ is used to describe the MDP process,

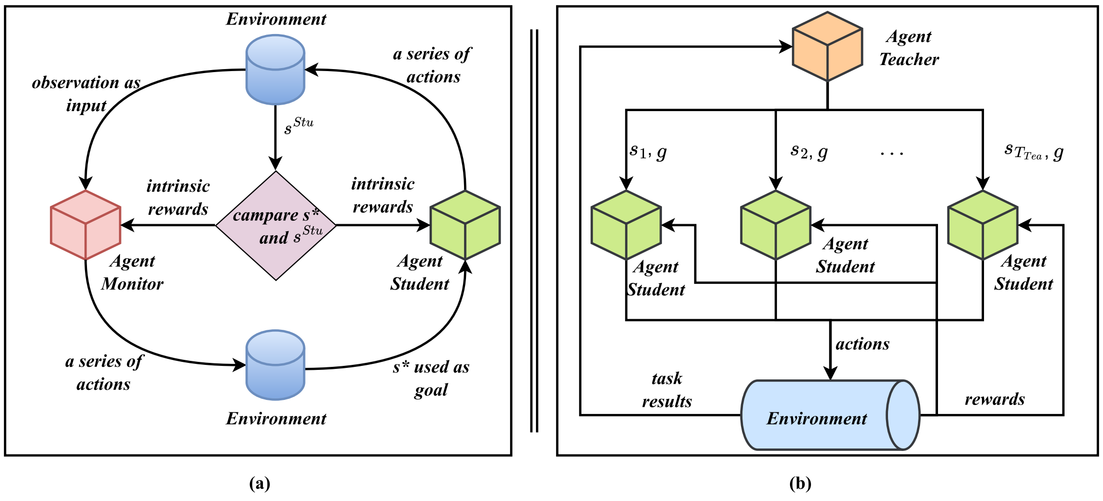


Fig. 2. Algorithm framework diagram. (a) Pretraining stage. (b) Fine-tuning stage.

where S is a finite set of states, A represents a finite set of actions, P denotes the probability of a state transition, R is a reward function, and γ is a discount factor used to calculate accumulated rewards. Through repeated interactions with the environment, the agent will gradually learn an optimal policy.

B. Hierarchical Reinforcement Learning

In classic RL, an agent's policy is optimized based on the reward obtained through continuous interaction. However, when the environment is complicated, the state space grows exponentially. This situation is the so-called curse of dimensionality. As a branch of RL, HRL [35], [36] divides a complex environment into several subtasks, and by solving each of the subproblems, the complex problem is finally solved.

Therefore, HRL has a high learning efficiency, avoids the curse of dimensionality, and solves the problem of a large state-action space. Moreover, it has better generalization ability compared with standard RL. However, HRL faces some challenges [37], such as policy instability, additional hyperparameters, and low sampling efficiency.

III. Model and Method

To solve the problem stated in Section I-B, an HGR is proposed in this work. In contrast to the traditional RL algorithms, the state relay mechanism is adopted in the game-playing process to temporarily retain the state information rather than resetting it after a task has been finished. Moreover, an auxiliary penalty is introduced to regulate the difficulty of tasks.

As Fig. 2 shows, the HGR contains two stages: the pretraining stage and the fine-tuning stage. The pretraining stage trains agent Student to explore the environment at a low level. The fine-tuning stage proposes subgoals for guiding Student's learning at a high level.

As depicted in Fig. 2(a), in the pretraining stage, there are two agents, monitor and student. The pretraining is an unsupervised two-player game process between monitor and student. Monitor first performs a sequence of actions, and the environment is ultimately transitioned to state s^*. Then, the environment state is reset to its initial state, and student begins to perform actions. The environment state is eventually transitioned to state s^{Stu} after student acts. If s^{Stu} is very close to s^*, student will receive an appropriate reward, and monitor receives no reward. Otherwise, student receives no reward, and monitor receives an appropriate reward. Through games playing with the monitor, the student can gradually explore and familiarize itself with the environment.

In the high-level fine-tuning stage, an additional agent teacher is included alongside student. Teacher proposes subgoals to guide student's learning. After the pretraining stage, student is already familiar with the environment and has the ability to complete some tasks. Therefore, student is more likely to achieve the subgoals proposed by teacher. As shown in Fig. 2(b), teacher proposes a new subgoal g every T_{Tea} time steps. When the environment state is at s_i, $i \in [1, 2, \ldots, T_{\text{Tea}}]$, student attempts to complete the subgoal g. Based on the results of Student in achieving the subgoal, teacher continuously fine-tunes the subgoal tasks such that student learns toward the target. Details of the pretraining and fine-tuning stages are presented below.

A. Pretraining Stage

Pretraining is performed via unsupervised training. Through the competitive game between monitor and student, the agent student can quickly become familiar with the environment. In the games, the monitor is responsible for proposing tasks; that is, the monitor first executes numerous actions, and then the final state of the environment after monitor acting is set as the target for the student to accomplish. If student completes the target, it will be rewarded 1, whereas monitor obtains a reward of 0. In contrast, if student cannot complete it, monitor will obtain a reward of 1. The monitor is allowed to propose tasks that are slightly higher than the student's ability to accelerate environment exploration.

The game starts with random initialization of monitor and student's policy: π_M and π_S. Both π_M and π_S take as input the current environmental state and output an action. The process of pretraining is described in Algorithm 1.

Algorithm 1 Pretraining Stage (T_M, T_S)

```
1:  // Initialization
2:  $t_M = 0$
3:  $i = 0$
4:  $s_0 = env.observe()$
5:  $s_i = s_0$
6:  while $k < K$ do
7:      $count = 0$
8:      while True do
9:          // Monitor's turn
10:         $t_M = t_M + 1$
11:         // Monitor's action
12:         $a = \pi_M(s_i)$
13:         if $a = STOP$ or $t_M \geq T_M$ then
14:             // Student's target state
15:             $s^* = s$
16:             $env.reset()$
17:             $break$
18:         end if
19:         $env.act(a)$
20:         $s_i = env.observe()$
21:     end while
22:     $t_S = 0$
23:     while True do
24:         // Student's turn
25:         set $s$ to $s_0$
26:         $s_i = s_0$
27:         // Student's action
28:         $a = \pi_S(s_i|g_t)$
29:         $env.act(a)$
30:         $s_i = env.observe()$
31:         $s^{Stu} = s_i$
32:         if $D\left(s^{Stu}, s^*\right) \leq \varepsilon$ and $t_S \leq T_S$ then
33:             // Student's reward
34:             $R_S = 1$
35:             $break$
36:         end if
37:         $t_S = t_S + 1$
38:     end while
39:     $s_k = s^{Stu}$
40:     // Monitor's reward
41:     $R_M = 1 - R_S$
42:     if $R_M = 1$ then
43:         $count = count + 1$
44:     end if
45:     if count $\geq n$ then
46:         // auxiliary penalty
47:         $R'_M = R_M - r$
48:         $count = 0$
49:     else
50:         $R'_M = R_M$
51:     end if
52:     if $R_S = 1$ then
53:         // Use the intermediate state
54:         $s_{k+1} = s_k$
55:     else
56:         // Reinitialize the state
            $s_{k+1}$ is reinitialized
57:     end if
58:     $k += 1$
59:     // Update the policy individually
60:     $\pi_M.update(R'_M)$
61:     $\pi_S.update(R_S)$
62: end while
63: $return$
```

First, monitor takes action according to its policy π_M, as shown in the following equation:

$$a_i^M = \pi_M(s_i). \tag{1}$$

In (1), s_i represents the state of the environment after monitor has taken ith steps. Monitor executes T_M steps in each game, so $i \in [0, 1, 2, \ldots, T_M]$. a_i^M represents the action taken by monitor when the environment is at state s_i. After monitor completes T_M steps, the state of the environment is denoted as s_t^M, which will be the target of student.

In student's turn, we use s_t^M as s^*, which is the target of student, and the environment state is then reset to its initial state s_0

$$a_i^S = \pi_S\left(s_i|s^*\right). \tag{2}$$

As shown in (2), given the target s^*, π_S takes as input the environment state s_i and outputs an action a_i^S for student. Student can execute T_S steps in each game, so $i \in [0, 1, 2, \ldots, T_S]$. After student completes T_S steps, the state of the environment is denoted as s^{Stu}.

Within the specified T_S steps, if student's final state s^{Stu} and the target state s^* satisfy the following equation, we deem that student completes the task, and thus, R_S is rewarded 1:

$$D\left(s^{\text{Stu}}, s^*\right) \leq \varepsilon. \tag{3}$$

In (3), D is a distance function and ε is a threshold, which is set according to the environment (see Section IV). If (3) is not satisfied within T_S, student fails and R_S is rewarded 0. Monitor's reward is $R_M = 1 - R_S$; that is, monitor and student compete for the reward. The reward is used to update monitor and student's policies π_M and π_S, respectively.

In this case, monitor will tend to seek a target state s^* that is difficult for student to complete. However, if the task is always too complicated, student will never be able to complete it, and thus, student cannot be trained effectively. To this end, an auxiliary penalty is introduced, where monitor is penalized if the proposed task is always too complicated

$$R'_M = R_M - r. \tag{4}$$

As shown in (4), r is a constant greater than 1, which is the auxiliary penalty. The reason for this choice is to ensure that the reward monitor receives is negative so that monitor adjusts its strategy. For example, if monitor continuously proposes n (e.g., $n = 4$ or 5) tasks that student cannot complete, monitor will be rewarded an additional -2 or more. In this way, monitor will appropriately adjust task difficulty. If student can complete the tasks assigned by monitor, monitor will attempt to find new tasks that have not been proposed to challenge student. Through this auxiliary penalty, monitor can propose various moderately difficult tasks, and student can effectively explore the environment.

In traditional RL training, regardless of whether the task is completed, the environment state in which the agent is situated will be reset after a game, which leads to the loss of useful information. Therefore, the state relay mechanism is proposed in this study to use the intermediate states of the training process fully

$$s_{k+1} = \begin{cases} s_k, & \text{if } k < K \\ \text{reset as } s_0, & \text{if } k > K, \text{ or task fails.} \end{cases} \tag{5}$$

In this mechanism, shown as (5), the condition for resetting the state is either: 1) it fails in one of the training tasks or

2) K tasks are all completed. Specifically, if student fails to complete a certain task, the algorithm remains the same; that is, the state is reset. However, if student accomplishes the task proposed by monitor, the current states in which monitor and student are situated will not be reset but rather used as the initial states in the next training game. When student successfully and consecutively completes multiple (e.g., K is three or four) tasks, the state is then reset to a random initial state. Through the relay mechanism, student can better explore the environment for a farther space that has not been visited.

B. Structure of the Student Agent

The student's policy is composed of two components. The first component is the target encoder E, which maps student's current state s_t^{Stu} and target state s^* to a low-dimensional target code $g_t \in \mathbb{R}^N$, as shown in the following equation:

$$g_t = E\left(s^*, s_t^{\mathrm{Stu}}\right). \tag{6}$$

Because of the low spatial dimension, the target encoder E can compactly express the student's target and retain sufficient task information.

The target encoder has two forms, as shown in the following equations, where ϕ is the state embedding function.

1) The first form calculates the difference between the current state and the target state

$$E\left(s^*, s_t^{\mathrm{Stu}}\right) = \phi\left(s^*\right) - \phi\left(s_t^{\mathrm{Stu}}\right). \tag{7}$$

2) The second form considers only the s^* state

$$E\left(s^*, s_t^{\mathrm{Stu}}\right) = \phi\left(s^*\right). \tag{8}$$

In the second part of student's policy, g_t is used as a parameter in π_S, as shown in the following equation:

$$a_S = \pi_S\left(s_t^{\mathrm{Stu}} | g_t\right). \tag{9}$$

Two components of Student's policy structure are then combined, where two parameters are used: the current state s_t^{Stu} and the target code g_t, as shown in the following equation:

$$a_S = \pi_S\left(s_t^{\mathrm{Stu}} | s^*\right) = \pi_S\left(s_t^{\mathrm{Stu}} | g_t\right) = \pi_S\left(s_t^{\mathrm{Stu}} | E\left(s^*, s_t^{\mathrm{Stu}}\right)\right). \tag{10}$$

C. Fine-Tuning Stage

After pretraining, student can be used as a low-level agent to complete more tasks. Then, the fine-tuning stage starts, the process of which is shown in Algorithm 2.

In this stage, an agent, named teacher with high-level policy π_T, is introduced to propose subgoals and encode them to generate g_t for student, as shown in the following equation:

$$g_t = \pi_T(s_t). \tag{11}$$

Then, the target code g_t is passed to the student's policy π_S, where it is converted into actions

$$a_{t+i} = \pi_S(s_{t+i} | g_t). \tag{12}$$

Student attempts to take actions to complete subgoal g_t when $i \in [0, 1, 2, \ldots, T_{\mathrm{Tea}} - 1]$ (T_{Tea} is the maximum step length of teacher). The time when teacher proposes a new subgoal is at $t + T_{\mathrm{Tea}}$. An external reward is used to fine-tune teacher's policy π_T and train student. After pretraining, student has been trained to complete the task with a total of T_M steps; here, we set $T_{\mathrm{Tea}} = T_M$.

Algorithm 2 Fine-Tuning Stage (T_{Tea}, T_S)

```
1: // Initialization
2: $i = 0, t_S = 0$
3: $R_T = 0,\ R_S = 0$
4: while $i < T_{Tea}$ do
5:     $s = env.observe()$
6:     // Teacher proposes subgoals
7:     $g_t = \pi_T(s)$
8:     while True do
9:         $s_i = env.observe()$
10:        // Student's action
11:        $a_i = \pi_S(s_i | g_t)$
12:        if $env.done()$ or $t_S \geq T_S$ then
13:            break
14:        end if
15:        $env.act(a_i)$
16:        // Student's reward
17:        $R_S = R_S + env.reward()$
18:        $t_S = t_S + 1$
19:    end while
20:    // Teacher's reward
21:    $R_T = R_T + env.reward()$
22:    $i = i + 1$
23: end while
24: // Update the policy individually
25: $\pi_T.update(R_T)$
26: $\pi_S.update(R_S)$
27: return
```

Note the difference between the two stages: in the pretraining stage, monitor's job is to propose a variety of tasks for student to complete. The key idea is that the game between them should help student understand what the environment is like and how to transition from one state to another so that it can learn the target task faster. In the fine-tuning stage, teacher's job is to propose appropriate subgoals to student based on the current task. When presented with a subgoal task, student tries to accomplish it (monitor plays no role in the fine-tuning stage).

D. Neural Network Modeling

The nonlinear multilayer perceptron (MLP) [38] is used to model π_M and π_S. The state embedding function ϕ is also a perceptron. π_M and ϕ have two layers, and the second layer of ϕ is set to be linear to avoid setting boundaries for the target. The policy π_S of the student is a three-layer perceptron, including a hidden layer in the middle, as shown in the following equation:

$$h_2 = f\left(W_2 f(W_1 s_t) + W_g g_t\right) \tag{13}$$

where f is the tanh activation function. To facilitate the experimental design, the bias term is omitted. The hidden layer includes 64 cells. All the policy nets have $D+1$ output heads, where D is the dimension of the action space.

The high-level policy π_T is modeled using the BP network. During training, error BP is used to make continuous corrections, thereby accelerating the convergence of the loss function. Moreover, the activation function is the rectified linear unit (ReLU) function, which reduces the interdependence between parameters, avoiding the occurrence of overfitting and effectively improving the training result.

IV. Experiment

The proposed HGR algorithm is expected to improve the performance compared with the state-of-the-art related HRL algorithms. Therefore, we use the SNN4HRL [14] algorithm, the HSP [30] algorithm, and the HRL with k-step adjacency constraint (HRAC) [20] algorithm as baselines. In our comparative experiments, the number of timesteps per iteration when training SNN4HRL is different from that in the original paper. We use the parameter setting of [39] to speed up SNN4HRL convergence.

These algorithms are tested in two public environments. The first test is in the "Key-Door" environment of the MazeBase platform [40]. Specifically, in the "Key-Door" environment, the agent finds the key to a door and then opens it to enter the other room for treasure hunting. The second test is in the AntGather environment of MuJoCo [41]. For AntGather, an ant-like robot collects targets of specific colors to obtain more rewards in an environment where objects are randomly placed.

In both the experiments, monitor's entropy regularization coefficient β is set to 0.01, and student's imitation coefficient α_1 is 0.03. We use the proximal policy optimization (PPO) algorithm [42] for training all the policies. For optimization using root mean square propagation (RMSProp), the learning rate lr is 0.001, α_2 is 0.98, and ϵ is $1e^{-8}$. Each experiment is run three times with different random seeds, and the mean value and standard deviation of the experimental results are reported.

A. MazeBase

In the "Key-Door" task of MazeBase [40], the environment is two rooms with a door, as shown in Fig. 3. The eventual goal for the agent is to reach the location of the treasure. To achieve this goal, the agent needs to first determine the location of the key, move to the location of the key, and pick it up. Then, the agent needs to search where the door is, move there to open the door, and finally enter the other room to find the treasure.

As shown in Fig. 3, the experiment involves three tasks, and the position of every element in the tasks is randomly generated in each game. We consider the different sizes of the three task environments, which are 7×7 and 9×9. Each task is tested three times independently for each map size. The mean and standard deviation of the test results are calculated. In the first task, the agent searches for the target [see Fig. 3 (left)]. A target is randomly placed in the room. If the agent finds the target and moves to the location, the task is completed. Task 1 is the simplest of the search tasks. In the second task [see Fig. 3 (middle)], two rooms are separated by a door. The agent has to learn to find the key in the room, and then find the door and open it. Compared with Task 1, Task 2 adds the task of picking up the key and needs to find the locations of the key and door. In the last task [see Fig. 3 (right)], the agent must first find the key in its room, open the door, and then go to the other room to reach the target location. This task is the most difficult.

If the agent completes the task, a reward of 1 is given; otherwise, the reward is 0. The randomness of the target location and the sparsity of the reward structure make the task challenging.

The observable value is a binary vector: map width $\times$ map height $\times$ word size. The vocabulary includes a description of objects, such as "agent," "door," "block," "key," and "goal." The possible actions include four single-step movements (up, down, right, left), pick up action, and stop action. The agent must choose the next action after the previous action. In the pretraining process, the $L2$ paradigm is used as the distance function D, and ε is set to 0; that is, the final state of student in the task must match the final state of monitor.

During pretraining, each game contains four rounds. In each round, monitor takes $T_M = 5$ steps one time, while student takes $T_S = 7$ steps to reach the target state. A discount ratio of 0.7 is used between training rounds, which is enough for monitor and student to pick up the key and enter the other room. Here, the target encoder only considers the target state. T_{Tea} is equal to T_M when training teacher. In pretraining, student is trained to accomplish the task with T_M steps.

Due to the critical importance of parameters T_M and T_S to the algorithm, their values are determined through experiments. We conducted an experiment to show the results of their sensitivity. Then, we selected appropriate values of T_M and T_S. The results are illustrated in Fig. 4 and Tables I–III. In the figures of all the experimental results, the X-axis is the number of training epochs; the Y-axis denotes the success rate, i.e., the ratio of the number of successfully completed tasks to the total tasks in each epoch. In the tables of all the experimental results, the maximum reward is the highest reward obtained during the completion of each task, and the average reward is calculated by dividing the total reward by the number of epochs.

According to Fig. 4 and Tables I–III, it can be observed that as T_M and T_S increase, the success rate of HGR improves more rapidly, and HGR also attains higher average and maximum rewards. The experimental results align with our expectations. Theoretically, a larger T_S allows student to explore a wider range of the environment and enhance its ability to accomplish tasks. As T_M and T_S increase, the performance of HGR will improve, but the time cost will also increase. To balance algorithm performance and time cost, we set $T_M = 5$, $T_S = 7$ in three tasks of MazeBase.

After determining the values of parameters T_M and T_S of HGR, we conducted experiments for the performance comparison of four algorithms. The experimental results are shown in Fig. 5. The green line is the result of our HGR algorithm. The results of SNN4HRL, HSP, and HRAC are marked in yellow, magenta, and spring-green, respectively.

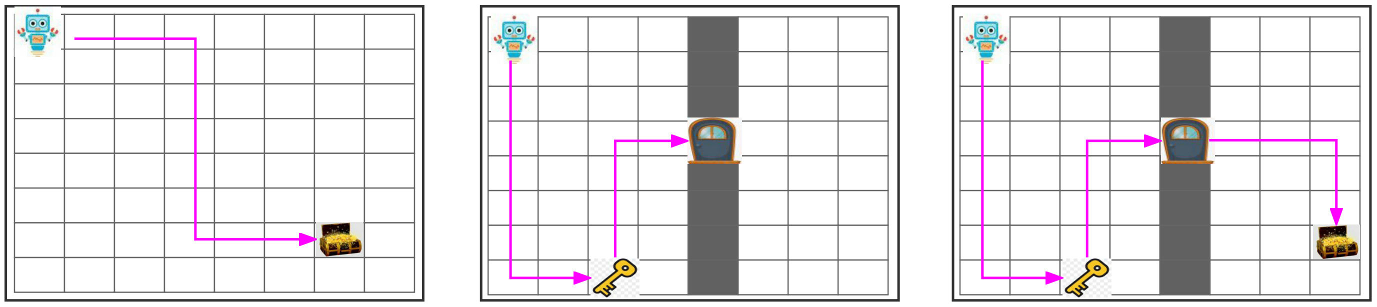


Fig. 3. MazeBase experimental environment.

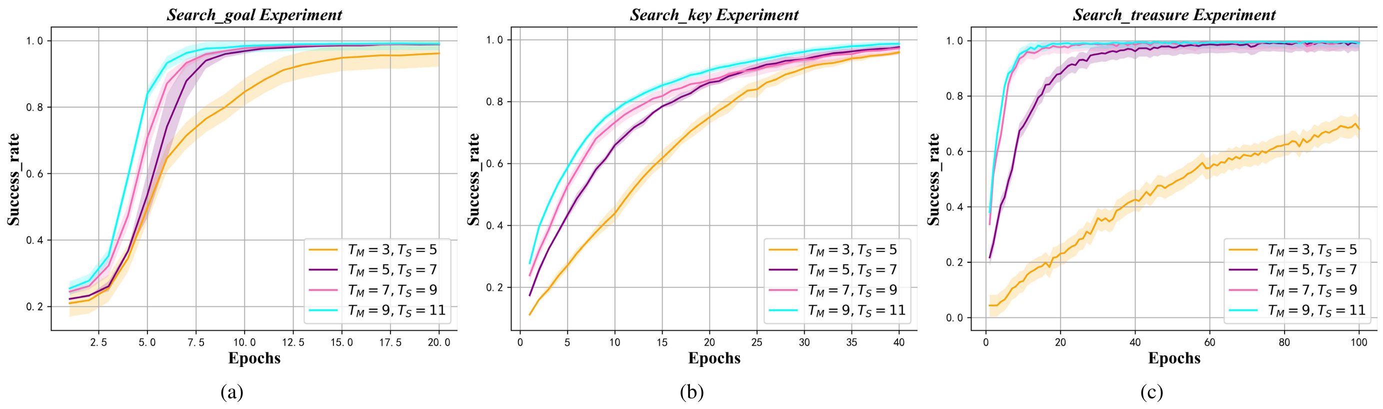


Fig. 4. Result comparison with different T_M and T_S under the MazeBase experiment. (a) Test results of Task 1. (b) Test results of Task 2. (c) Test results of Task 3.

TABLE I
REWARD COMPARISON WITH DIFFERENT T_M AND T_S IN TASK 1 OF MAZEBASE

Parameter value	Maximum reward	Average reward
HGR($T_M = 3, T_S = 5$)	5,538	2,645
HGR($T_M = 5, T_S = 7$)	5,671	2,784
HGR($T_M = 7, T_S = 9$)	5,697	2,848
HGR($T_M = 9, T_S = 11$)	5,706	2,872

TABLE II
REWARD COMPARISON WITH DIFFERENT T_M AND T_S IN TASK 2 OF MAZEBASE

Parameter value	Maximum reward	Average reward
HGR($T_M = 3, T_S = 5$)	10,184	4,534
HGR($T_M = 5, T_S = 7$)	10,367	4,933
HGR($T_M = 7, T_S = 9$)	10,322	4,993
HGR($T_M = 9, T_S = 11$)	10,477	5,079

Due to the simplicity of Task 1, all these algorithms can train the agents to gain a high success rate. However, the SNN4HRL [14] algorithm is much lower than HSP [30], HRAC [20], and our HGR. During training, HRAC's initial success rate is lower than that of HSP but later exceeds that of HSP and eventually approaches that of HGR. Similarly, in Tasks 2 and 3, HRAC's initial success rate is lowest but later exceeds SNN4HRL and HSP, eventually approaching HGR. For the second task, the convergence speed and success rate of the SNN4HRL algorithm are lower than those of HSP. For the most difficult task, Task 3, the initial convergence of SNN4HRL is lower than that of HSP but eventually slightly

TABLE III
REWARD COMPARISON WITH DIFFERENT T_M AND T_S IN TASK 3 OF MAZEBASE

Parameter value	Maximum reward	Average reward
HGR($T_M = 3, T_S = 5$)	20,528	9,131
HGR($T_M = 5, T_S = 7$)	29,168	15,973
HGR($T_M = 7, T_S = 9$)	29,194	16,224
HGR($T_M = 9, T_S = 11$)	29,244	16,268

exceeds that of HSP. Moreover, the convergence speed and success rate of SNN4HRL, HSP, and HRAC are significantly lower than those of HGR, indicating that our proposed method can be used to improve the performance of the state-of-the-art HRL algorithm even in complex tasks.

Relative experiments are conducted separately to show the effects of the relay mechanism and the auxiliary penalty mechanism. The results are shown in Fig. 6. The magenta line is the result of the baseline algorithm HSP [30]. The red line is the result of the auxiliary penalty mechanism (Only + Auxiliary Penalty), and the black line is that of the relay mechanism (Only + Relay). The dodger blue line is the joint effect of the two mechanisms (Auxiliary Penalty + Relay).

For Task 1, the convergence speed of the auxiliary penalty mechanism is slightly superior to that of the relay mechanism. For Task 2, the effects of the two mechanisms are similar. For Task 3, the incipient training is faster for the auxiliary penalty mechanism; however, after 20 epochs, the effect of the relay mechanism is improved. In addition, the best overall result can be seen after using both the mechanisms rather than each mechanism alone, and the success rate is also the highest,

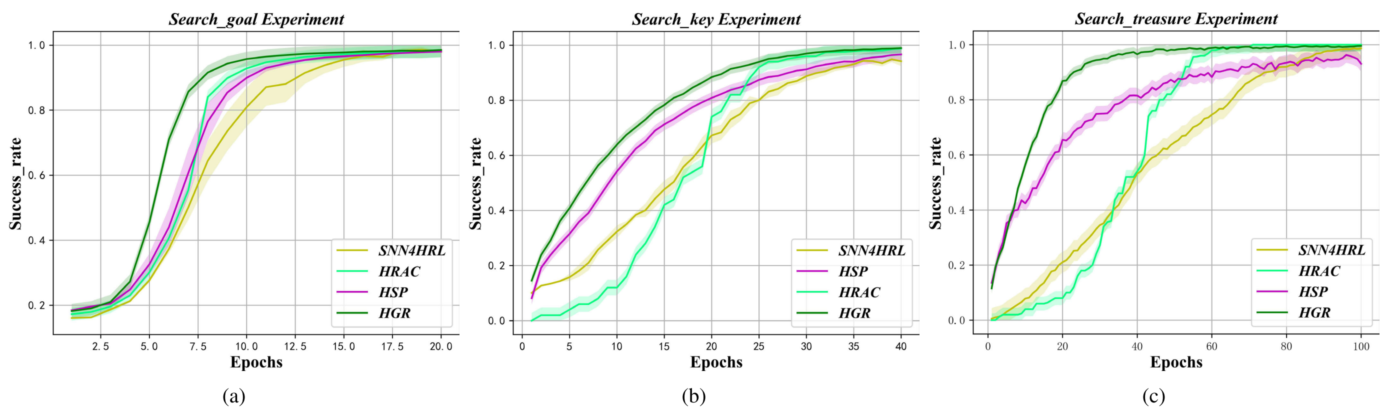


Fig. 5. Algorithm comparison under the MazeBase experiment. (a) Test results of Task 1. (b) Test results of Task 2. (c) Test results of Task 3.

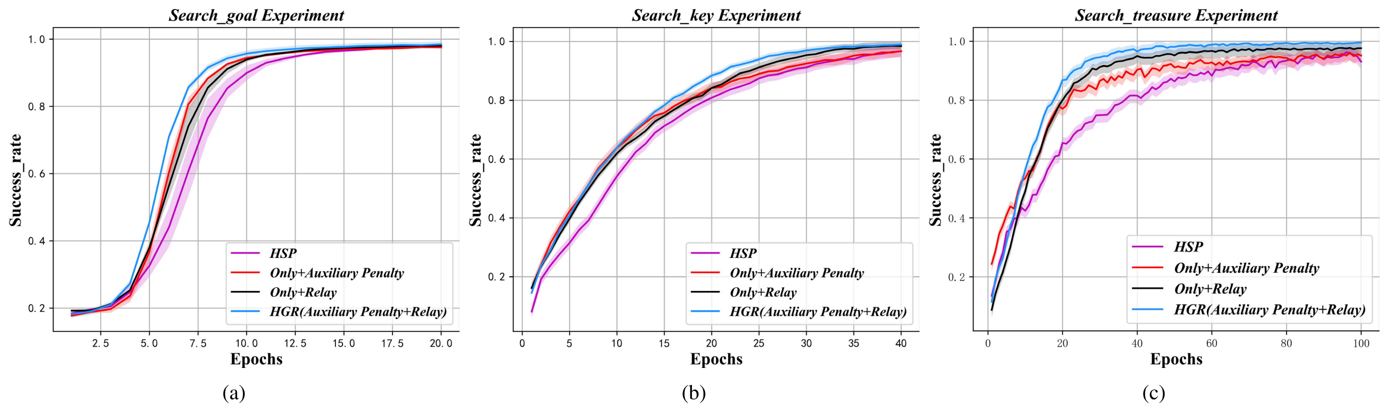


Fig. 6. Comparison of different mechanisms of the HGR algorithm under the MazeBase experiment. (a) Test results of Task 1. (b) Test results of Task 2. (c) Test results of Task 3.

TABLE IV
REWARD COMPARISON IN TASK 1 OF MAZEBASE

Algorithm	Maximum reward	Average reward
SNN4HRL	5,618	2,597
HSP	5,640	2,686
HRAC	5,657	2,711
HGR	5,671	2,784

TABLE V
REWARD COMPARISON IN TASK 2 OF MAZEBASE

Algorithm	Maximum reward	Average reward
SNN4HRL	9,996	4,290
HSP	10,258	4,689
HRAC	10,482	4,465
HGR	10,503	4,933

TABLE VI
REWARD COMPARISON IN TASK 3 OF MAZEBASE

Algorithm	Maximum reward	Average reward
SNN4HRL	28,912	12,798
HSP	27,265	14,539
HRAC	29,001	14,295
HGR	29,168	15,973

indicating the joint effect of these two mechanisms on the overall performance.

To further show the performance of the proposed algorithm, we report the maximum reward and average reward that the training agent receives in Tasks 1–3 of MazeBase in Tables IV–IX.

Based on Tables IV–VI, SNN4HRL's maximum reward exceeds that of HSP only in Task 3. In other tasks, SNN4HRL has the lowest average reward and maximum reward. In the three tasks, HRAC's maximum reward is higher than that of HSP and SNN4HRL. HSP's average reward is slightly lower than that of HRAC only in Task 1. In other tasks, HSP's average reward is higher than HRAC and SNN4HRL. In all the tasks, HGR has the highest value in maximum and average reward, which demonstrates the excellent ability of HGR to complete tasks.

According to Tables VII–IX, overall, the average and maximum rewards of the relay mechanism are higher than those of the auxiliary penalty mechanism and most baselines. This proves that the relay mechanism can make full use of environmental information and play an important role in improving the performance of HGR. The average reward and maximum reward of the auxiliary penalty mechanism are higher than those of HSP and SNN4HRL. This shows that the auxiliary penalty mechanism is also a useful part of our algorithm. HGR endowed with these two mechanisms achieves the highest maximum reward and the highest average reward.

TABLE VII
REWARD COMPARISON OF ABLATION EXPERIMENT IN TASK 1 OF MAZEBASE

Algorithm	Maximum reward	Average reward
Only+Auxiliary Penalty	5,643	2,746
Only+Relay	5,648	2,744
HGR(without fine-tuning stage)	1,765	820
HGR(without pretraining stage)	3,166	1,551
HGR	5,671	2,784

TABLE VIII
REWARD COMPARISON OF ABLATION EXPERIMENT IN TASK 2 OF MAZEBASE

Algorithm	Maximum reward	Average reward
Only+Auxiliary Penalty	10,264	4,804
Only+Relay	10,312	4,891
HGR(without fine-tuning stage)	3,432	1,660
HGR(without pretraining stage)	8,611	3,142
HGR	10,367	4,933

TABLE IX
REWARD COMPARISON OF ABLATION EXPERIMENT IN TASK 3 OF MAZEBASE

Algorithm	Maximum reward	Average reward
Only+Auxiliary Penalty	27,876	15,059
Only+Relay	28,607	15,589
HGR(without fine-tuning stage)	6,153	3,534
HGR(without pretraining stage)	19,398	8,586
HGR	29,168	15,973

To demonstrate the impact of HGR's different stages on the experimental results, we conducted one additional ablation experiment. The results are illustrated in Fig. 7 and Tables VII–IX.

Fig. 7 and Tables VII–IX show that HGR (without fine-tuning stage) has the worst performance, which shows that the high-level strategy of the hierarchical structure plays an important role in HGR. In addition, the HGR (without the pretraining stage) success rate is also slower than that of HGR. This shows that the game process truly helps student explore the environment. This ablation experiment proves that hierarchical structure and playing games between agents are important and take a key role in the effectiveness of HGR.

B. MuJoCo

The algorithms are then evaluated on the MuJoCo [41] platform. The environment of the AntGather in MuJoCo is shown in Fig. 8.

The agent is an ant-like robot with four legs, which is placed in a coliseum with green and red objects. All the object positions are random in each round. If the ant robot touches a green object, the reward value is $+1$; if it is red, the reward is -1. If the agent jumps up in the Z-axis direction, it will be penalized by -10, and the round will terminate immediately. This setting increases the difficulty of the task because it inhibits the search and possibly introduces a local minimum. The ant agent must learn to stay still to avoid the penalty. Each round ends after a thousand steps.

A 125-D vector, including position, velocity, and joint angles, is taken as the observation value. The action of the agent is an 8-D vector, i.e., the movements of the eight joints of the ant. Although the actions are continuous, they are discretized into five 2-D vectors. Here, we use the difference version of the target encoder, as shown in (7). The distance function D is defined as the physical distance on the X–Y-axis, and $\epsilon = 0.25$. This means that the task is considered completed if student's final location is less than 0.25 from monitor's last location. During pretraining, each game contains four rounds. In each round, monitor takes $T_M = 50$ steps as a series of actions, and student takes $T_S = 70$ steps. In the fine-tuning stage, when training teacher, we also set $T_{\text{Tea}} = T_M$. An entire action of the teacher also consists of 50 steps, and it can take 20 actions in a round.

The values of T_M and T_S are also determined through experiments. We conducted an experiment to show the results of their sensitivity in AntGather. Then, we selected appropriate values of T_M and T_S. Each experiment is also run three times independently, and the mean of the experimental results is calculated, as illustrated in Fig. 9 and Table X. In the figures of all the experimental results, the X-axis is the number of training epochs; the Y-axis is the reward the agent obtains during training. In the tables of all the experimental results, the maximum reward is the highest reward obtained during the completion of each task, and the average reward is calculated by dividing the total reward by the number of epochs.

According to Fig. 9 and Table X, it can be observed that the general trend is still that as T_M and T_S increase, the reward of HGR improves more rapidly and attains higher average and maximum rewards. Although the maximum reward of HGR ($T_M = 70, T_S = 90$) is very close to that of HGR ($T_M = 90, T_S = 110$), the average reward of HGR ($T_M = 70, T_S = 90$) is 0.34 lower than that of HGR ($T_M = 90, T_S = 110$). Overall, student performs more steps, which is helpful for exploring the environment and learning policies. As T_M and T_S increase, the time cost will also increase. To balance algorithm performance and time cost, we set $T_M = 50, T_S = 70$ in the AntGather task of MuJoCo.

After determining the values of parameters T_M and T_S of HGR in AntGather, we conducted experiments for the performance comparison of four algorithms. The experimental results are shown in Fig. 10. The green line is the result of the HGR algorithm proposed in this work. The magenta line is the result of the HSP algorithm [30]. The yellow line is the SNN4HRL algorithm [14]. The spring-green line is the HRAC algorithm [20]. Since AntGather is a challenging task, it can be seen from Fig. 10 that for the SNN4HRL algorithm, the reward converges to 0 after training for a long time, indicating that the agent cannot learn the correct policy. For the HSP algorithm [30], the reward value reaches more than 1.0, which indicates the effectiveness of the combination of hierarchical learning and self-play.

The HRAC alleviates the goal space by restricting the high-level action space from the whole goal space to a k-step adjacent region of the current state using an adjacency constraint. The agent can gradually learn useful policies by effective subgoals proposed by the high level and obtain more

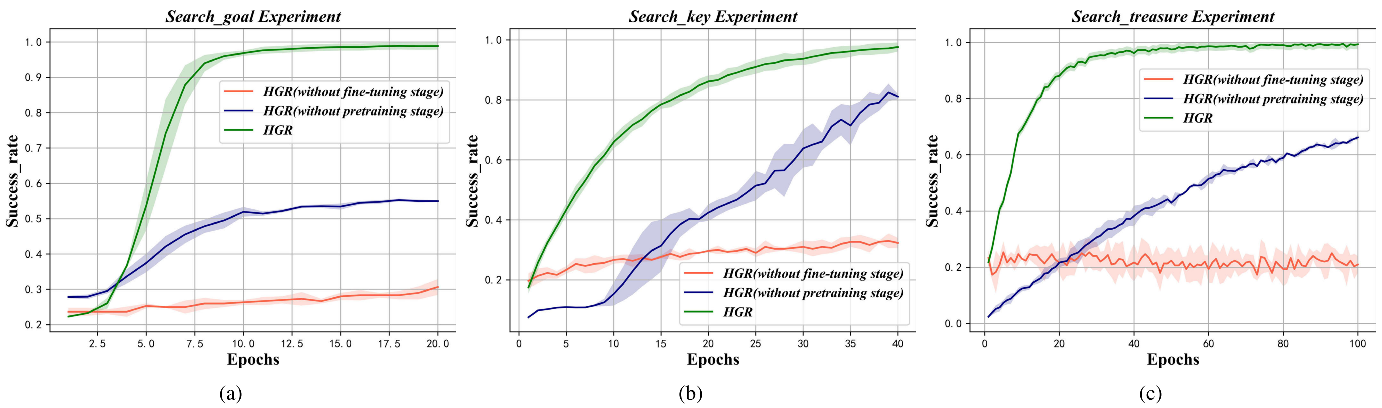


Fig. 7. Impact of HGR's different stages on experimental results under the MazeBase experiment. (a) Test results of Task 1. (b) Test results of Task 2. (c) Test results of Task 3.

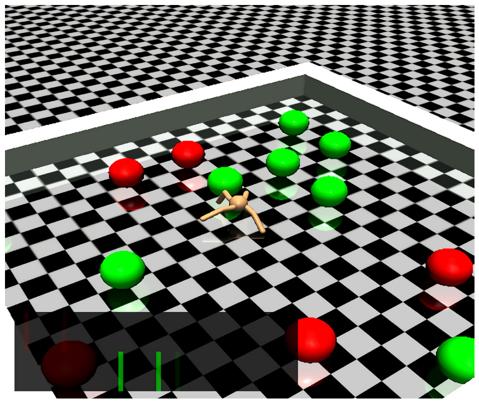


Fig. 8. AntGather experiment environment.

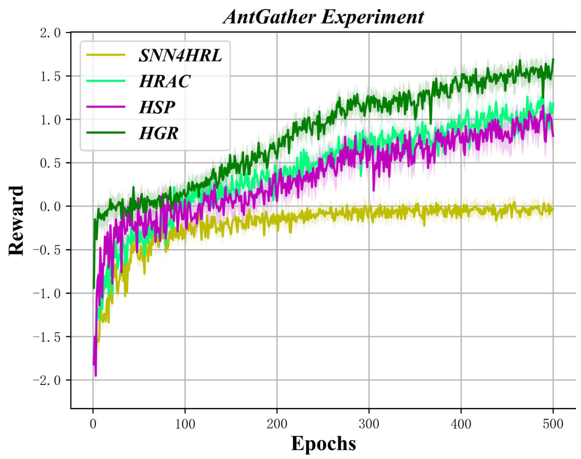


Fig. 10. Algorithm comparison under the AntGather experiment.

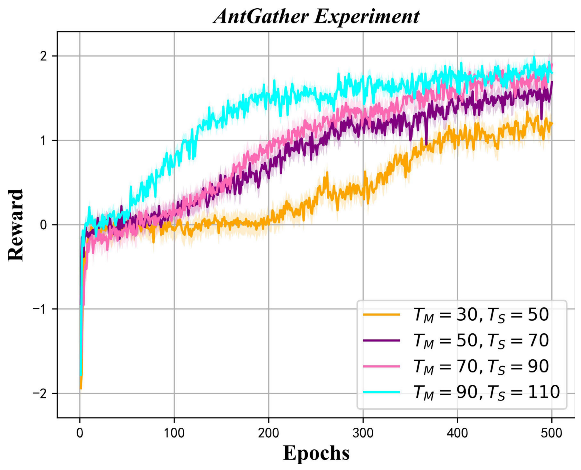


Fig. 9. Result comparison with different T_M and T_S under the AntGather experiment.

TABLE X
REWARD COMPARISON WITH DIFFERENT T_M AND T_S IN ANTGATHER EXPERIMENT

Parameter value	Maximum reward	Average reward
HGR($T_M = 30, T_S = 50$)	1.34	0.42
HGR($T_M = 50, T_S = 70$)	1.69	0.83
HGR($T_M = 70, T_S = 90$)	1.90	0.95
HGR($T_M = 90, T_S = 110$)	1.99	1.29

TABLE XI
REWARD COMPARISON IN THE ANTGATHER EXPERIMENT

Algorithm	Maximum reward	Average reward
SNN4HRL	0.048	-0.215
HSP	1.16	0.343
HRAC	1.30	0.419
HGR	1.69	0.832

rewards (0.14) than HSP, but the performance is still no better than HGR. Regarding the HGR algorithm, its reward is the highest (i.e., 1.69).

The reward value in the AntGather experiment is reported in Table XI. For SNN4HRL, both the maximum reward and average reward are inferior to those of the other algorithms. The HGR's maximum reward and the average reward are also the highest among the baselines. These results further indicate that HGR can advance the performance of the state-of-the-art HRL algorithms in complex environments.

To verify the effects of these two mechanisms proposed in HGR, each mechanism is tested separately. The results are shown in Fig. 11. The magenta line is the result of the baseline algorithm HSP [30]. The red line is the result of the auxiliary

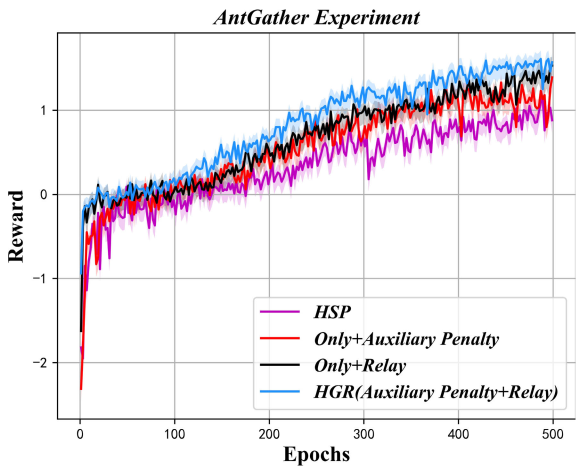


Fig. 11. Comparison of different mechanisms of the HGR algorithm under the AntGather experiment.

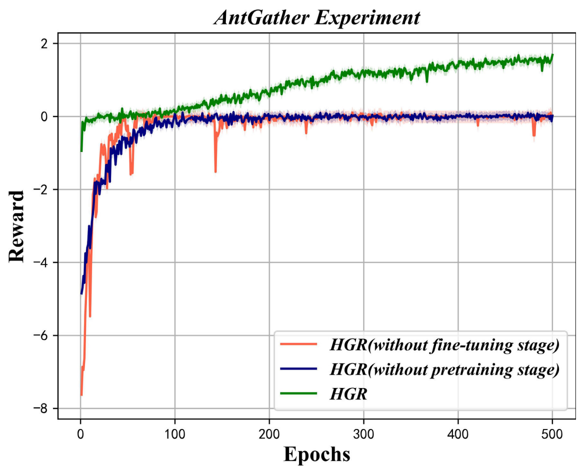


Fig. 12. Impact of HGR's different stages on the experimental results of the AntGather experiment.

TABLE XII
REWARD COMPARISON OF THE ANTGATHER ABLATION EXPERIMENT

Algorithm	Maximum reward	Average reward
Only+Auxiliary Penalty	1.39	0.559
Only+Relay	1.53	0.655
HGR(without fine-tuning stage)	0.11	-0.240
HGR(without pretraining stage)	0.12	-0.221
HGR	1.69	0.832

penalty mechanism (Only + Auxiliary Penalty), and the black line is that of the relay mechanism (Only + Relay). The dodger blue line is the joint effect of the two mechanisms (Auxiliary Penalty + Relay).

As Fig. 11 and Table XII show, introducing the auxiliary penalty mechanism improves the stability of the training process and reduces the fluctuation of rewards received during training compared with HSP [30]. The final reward is approximately 1.39, which is also higher than HSP. In comparison, the relay mechanism gains better training results, and the reward value reaches approximately 1.53. By integrating the two mechanisms, the final reward value obtained by the HGR algorithm reaches 1.69 or more, further indicating the effectiveness of the two proposed mechanisms and their joint effect in the HGR algorithm.

To demonstrate the impact of HGR's different stages on the experimental results, we conducted one additional ablation experiment in AntGather. The results are illustrated in Fig. 12 and Table XII.

According to Fig. 12 and Table XII, although the average and maximum rewards of HGR (without the pretraining stage) are slightly higher than those of HGR (without the fine-tuning stage), each performance is much worse than that of HGR. This shows that hierarchical structure and playing games between agents are both indispensable and take a key part in the effectiveness of HGR.

V. CONCLUSION

We proposed an HRL algorithm for motion planning based on the state relay mechanism and the auxiliary penalty mechanism. The first stage of the algorithm does not require external environment incentives and only relies on the game between two agents. One of these agents continues proposing new tasks while the other agent attempts to complete the tasks. The auxiliary penalty helps adjust the difficulty of the tasks proposed by the first agent, and the relay mechanism helps improve the sample utilization efficiency and enables the second agent to explore more information about the environment. Therefore, after training, we obtain an agent who is very familiar with the environment.

In the second stage, a high-level policy guides the agent obtained in the first stage, and the BP neural network is used to train the high-level policy to propose appropriate subgoals. The hierarchical structure is suitable for complex environments with sparse rewards, such as AntGather.

In the pretraining process of the proposed algorithm, the game is played between two agents. In future studies, we will consider the case of games played among multiple agents [43], which enables parallel learning of multiple agents and allows the Student to learn more knowledge within the same amount of time, thereby accelerating the environment exploration process.

REFERENCES

[1] X. Wang et al., "Deep reinforcement learning: A survey," *IEEE Trans. Neural Netw. Learn. Syst*, vol. 35, no. 4, pp. 5064–5078, Apr. 2024.
[2] S. Wen, Z. Wen, D. Zhang, H. Zhang, and T. Wang, "A multi-robot path-planning algorithm for autonomous navigation using meta-reinforcement learning based on transfer learning," *Appl. Soft Comput.*, vol. 110, Oct. 2021, Art. no. 107605.
[3] E. Zhao, N. Zhou, C. Liu, H. Su, Y. Liu, and J. Cong, "Time-aware MADDPG with LSTM for multi-agent obstacle avoidance: A comparative study," *Complex Intell. Syst.*, pp. 1–15, Mar. 2024, doi: 10.1007/s40747-024-01389-0.
[4] Y. Jin, S. Wei, J. Yuan, and X. Zhang, "Hierarchical and stable multiagent reinforcement learning for cooperative navigation control," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 1, pp. 90–103, Jan. 2023.
[5] D. Silver et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.
[6] Y. Wu, S. Liao, X. Liu, Z. Li, and R. Lu, "Deep reinforcement learning on autonomous driving policy with auxiliary critic network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 7, pp. 3680–3690, Jul. 2023.

[7] R. Parr and S. Russell, “Reinforcement learning with hierarchies of machines,” in *Proc. Conf. Adv. Neural Inf. Process. Syst.*, 1997, pp. 1043–1049.

[8] R. S. Sutton, D. Precup, and S. Singh, “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,” *Artif. Intell.*, vol. 112, nos. 1–2, pp. 181–211, Aug. 1999.

[9] T. G. Dietterich, “Hierarchical reinforcement learning with the MAXQ value function decomposition,” *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, Nov. 2000.

[10] J. Cong, Y. Liu, and C. Liu, “Guiding task learning by hierarchical rl with an experience replay mechanism through reward machines,” in *Proc. PRICAI*, 2024, pp. 164–170.

[11] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 3682–3690.

[12] V. Mnih, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[13] A. S. Vezhnevets et al., “Feudal networks for hierarchical reinforcement learning,” in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 3540–3549.

[14] C. Florensa, Y. Duan, and P. Abbeel, “Stochastic neural networks for hierarchical reinforcement learning,” in *Proc. 5th Int. Conf. Learn. Represent.*, Toulon, France, 2017, pp. 1–17.

[15] M. Andrychowicz et al., “Hindsight experience replay,” 2017, *arXiv:1707.01495*.

[16] L. Schramm, Y. Deng, E. Granados, and A. Boularias, “USHER: Unbiased sampling for hindsight experience replay,” in *Proc. Conf. Robot Learn.*, vol. 205, Auckland, New Zealand, Dec. 2022, pp. 2073–2082.

[17] L. Moro, A. Likmeta, E. Prati, and M. Restelli, “Goal-directed planning via hindsight experience replay,” in *Proc. 10th Int. Conf. Learn. Represent.*, 2022, pp. 1–16.

[18] A. Demin and D. Ponomaryov, “Interpretable reinforcement learning with multilevel subgoal discovery,” in *Proc. 21st IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2022, pp. 251–258.

[19] S. Pateria, B. Subagdja, A. Tan, and C. Quek, “End-to-end hierarchical reinforcement learning with integrated subgoal discovery,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 12, pp. 7778–7790, Dec. 2022.

[20] T. Zhang, S. Guo, T. Tan, X. Hu, and F. Chen, “Adjacency constraint for efficient hierarchical reinforcement learning,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 4, pp. 4152–4166, Apr. 2023.

[21] C. Liu, E. Zhu, Q. Zhang, and X. Wei, “Exploring the effects of computational costs in extensive games via modeling and simulation,” *Int. J. Intell. Syst.*, vol. 36, no. 8, pp. 4065–4087, Aug. 2021.

[22] J. R. H. Mariño and C. F. M. Toledo, “Evolving interpretable strategies for zero-sum games,” *Appl. Soft Comput.*, vol. 122, Jun. 2022, Art. no. 108860.

[23] M. Li, J. Qin, Q. Ma, W. X. Zheng, and Y. Kang, “Hierarchical optimal synchronization for linear systems via reinforcement learning: A Stackelberg-Nash game perspective,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 4, pp. 1600–1611, Apr. 2021.

[24] B. M. Albaba and Y. Yildiz, “Driver modeling through deep reinforcement learning and behavioral game theory,” *IEEE Trans. Control Syst. Technol.*, vol. 30, no. 2, pp. 885–892, Mar. 2022.

[25] A. L. Samuel, “Some studies in machine learning using the game of checkers,” *IBM J. Res. Develop.*, vol. 3, no. 3, pp. 210–229, Jul. 1959.

[26] G. Tesauro, “Temporal difference learning and TD-Gammon,” *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.

[27] D. Silver et al., “Mastering the game of go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.

[28] J. Schrittwieser et al., “Mastering atari, go, chess and shogi by planning with a learned model,” *Nature*, vol. 588, no. 7839, pp. 604–609, Dec. 2020.

[29] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus, “Intrinsic motivation and automatic curricula via asymmetric self-play,” in *Proc. 6th Int. Conf. Learn. Represent.*, Vancouver, BC, Canada, 2018, pp. 1–16.

[30] S. Sukhbaatar, E. Denton, A. Szlam, and R. Fergus, “Learning goal embeddings via self-play for hierarchical reinforcement learning,” 2018, *arXiv:1811.09083*.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[32] M.-P. Song, G.-C. Gu, and G.-Y. Zhang, “Survey of multi-agent reinforcement learning in Markov games,” *Control Decis.*, vol. 20, no. 10, p. 1081, 2005.

[33] C. Florensa, D. Held, X. Geng, and P. Abbeel, “Automatic goal generation for reinforcement learning agents,” in *Proc. Int. Conf. Mach. Learn.*, vol. 80, 2018, pp. 1515–1528.

[34] C. Florensa, D. Held, M. Wulfmeier, and P. Abbeel, “Reverse curriculum generation for reinforcement learning,” in *Proc. Int. Conf. Robot Learn.*, vol. 78, 2017, pp. 482–495.

[35] S. Pateria, B. Subagdja, A.-H. Tan, and C. Quek, “Hierarchical reinforcement learning: A comprehensive survey,” *ACM Comput. Surv.*, vol. 54, no. 5, pp. 1–35, 2021.

[36] Z. Cao and C.-T. Lin, “Reinforcement learning from hierarchical critics,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 2, pp. 1066–1073, Feb. 2023.

[37] J. Lai, J. Y. Wei, and X. L. Chen, “Overview of hierarchical reinforcement learning,” *Comput. Eng. Appl.*, vol. 57, no. 3, pp. 72–79, 2021.

[38] A. Pinkus, “Approximation theory of the MLP model in neural networks,” *Acta Numerica*, vol. 8, pp. 143–195, Jan. 1999.

[39] S. Li, R. Wang, M. Tang, and C. Zhang, “Hierarchical reinforcement learning with advantage-based auxiliary rewards,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, Vancouver, BC, Canada, 2019, pp. 1407–1417.

[40] S. Sukhbaatar, A. Szlam, G. Synnaeve, S. Chintala, and R. Fergus, “MazeBase: A sandbox for learning from games,” 2015, *arXiv:1511.07401*.

[41] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: A physics engine for model-based control,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 5026–5033.

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017, *arXiv:1707.06347*.

[43] C. Liu, E. Zhu, Q. Zhang, and X. Wei, “Modeling of agent cognition in extensive games via artificial neural networks,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4857–4868, Oct. 2018.

Chanjuan Liu (Member, IEEE) received the B.E. degree in software engineering from Jinan University, Guangzhou, China, in 2010, and the Ph.D. degree in computer software and theory from Peking University, Beijing, China, in 2016.

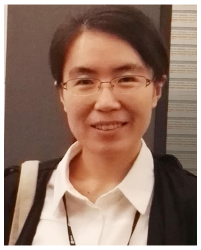
She is currently an Associate Professor with the School of Computer Science and Technology, Dalian University of Technology, Dalian, China. Her current research interests include deep learning and game theory.

Jinmiao Cong received the B.E. degree in computer science and technology from Dalian University of Technology, Dalian, China, in 2021, where he is currently pursuing the master’s degree in computer science and technology.

His research interests include deep reinforcement learning and multiagent systems.

Guangyuan Liu received the B.E. degree in computer science and technology from Northeastern University, Shenyang, China, in 2020. He is currently pursuing the master's degree in computer science and technology with Dalian University of Technology, Dalian, China.

His research interests include deep learning and combinatorial optimization.



Guifei Jiang received the double Ph.D. degree in computer science from Western Sydney University, Penrith, NSW, Australia, and the University of Toulouse, Toulouse, France, in 2016.

She is currently an Associate Professor with Nankai University, Tianjin, China. Her research interests include general game playing and multi-agent systems.

Xirong Xu received the B.E. degree in applied mathematics from the East China University of Science and Technology, Shanghai, China, in 1990, and the M.E. and Ph.D. degrees in computer software and theory from Dalian University of Technology, Dalian, China, in 2002 and 2005, respectively.

She is currently an Associate Professor with the School of Computer Science and Technology, Dalian University of Technology. Her current research interest is combinatorial optimization.

Enqiang Zhu received the B.E. degree in information and computing science and the M.E. degree in operational research and cybernetics from Lanzhou Jiaotong University, Lanzhou, China, in 2007 and 2010, respectively, and the Ph.D. degree in computer science from Peking University, Beijing, China, in 2015.

He is currently a Professor with the Institute of Computing Science and Technology, Guangzhou University, Guangzhou, China. His current research interests include graph theory and machine learning.