Autonomous Agents and Multi-Agent Systems (2023) 37:14
https://doi.org/10.1007/s10458-022-09595-1

Learning by reusing previous advice: a memory-based teacher–student framework

Changxi Zhu[1] · **Yi Cai**[1] · **Shuyue Hu**[2] · **Ho-fung Leung**[3] · **Dickson K. W. Chiu**[4]

Accepted: 10 December 2022 / Published online: 29 December 2022

Abstract

Reinforcement Learning (RL) has been widely used to solve sequential decision-making problems. However, it often suffers from slow learning speed in complex scenarios. Teacher–student frameworks address this issue by enabling agents to ask for and give advice so that a student agent can leverage the knowledge of a teacher agent to facilitate its learning. In this paper, we consider the effect of reusing previous advice, and propose a novel memory-based teacher–student framework such that student agents can memorize and reuse the previous advice from teacher agents. In particular, we propose two methods to decide whether previous advice should be reused: *Q-Change per Step* that reuses the advice if it leads to an increase in Q-values, and *Decay Reusing Probability* that reuses the advice with a decaying probability. The experiments on diverse RL tasks (Mario, Predator–Prey and Half Field Offense) confirm that our proposed framework significantly outperforms the existing frameworks in which previous advice is not reused.

Keywords Reinforcement learning · Multi-agent learning · Action advising · Teacher–student

✉ Shuyue Hu
hushuyue@pjlab.org.cn

Changxi Zhu
c.zhu@uu.nl

Yi Cai
ycai@scut.edu.cn

Ho-fung Leung
lhf@cuhk.edu.hk

Dickson K. W. Chiu
dicksonchiu@ieee.org

[1] School of Software Engineering, South China University of Technology, Guangzhou, China

[2] Shanghai Artificial Intelligence Laboratory, Shanghai, China

[3] Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China

[4] Faculty of Education, The University of Hong Kong, Hong Kong, China

1 Introduction

Reinforcement learning (RL) has been employed to solve many real-world problems, e.g., robotics, optimizing memory control, and personalized web services [3, 16]. However, RL often suffers from slow learning speed in complex applications. This can be further intensified when multiple agents are independently learning and observing since they adapt to one another. Undoubtedly, when a trained agent or a human expert is available, a new agent can benefit from asking advice for the current task. Even in a more general situation where all agents learn without prior knowledge, they can accelerate learning process by sharing acquired knowledge. Under practical constraints of limited computing resources or communication, agents need to decide what and when to share, as well as how to utilize the shared knowledge.

Recently, the teacher–student framework [25] has received much attention. In this paradigm [25], a trained agent (named teacher) advises a learning agent (named student) on which action to take in a state. Teachers and students are only required to have the same action set without specifying their learning structures. This provides much flexibility for practical scenarios in which agents may be equipped with different sensors or policy representations to collaborate. Silva et al. [9] focus on how action advising improves agents' mutual learning processes, and adapt the teacher–student framework to multi-agent settings where multiple simultaneously learning agents exist. In particular, advising opportunities are established on-demand taking into account the communication cost among agents. Omidshafiei et al. [19] view teaching in Multi-Agent Reinforcement Learning (MARL) as a high-level sequential decision task. Agents taking a student's role learn to ask for advice or not, while agents taking a teacher's role learn to advise heterogeneous teammates for actions.

Previous works on action advising focus on the problems of when and what to advise, but rarely on the problem of how to use the advice more efficiently. A key assumption [2, 9, 25] behind the teacher–student framework and its variations is that the teachers are more experienced than the students. However, the advice from the teachers is performed only once and then forgotten. We conjecture that reusing the advice, advice will improve the efficiency of learning. Imagine that a coach teaches a rookie how to shoot in a soccer game, and the rookie is instructed to aim following his coach's advice. However, he may miss the shot at this time due to a noisy and stochastic environment, even though the action advised by the coach is optimal. When the rookie tries to shoot again, if he overlooks the previous advice, then it is likely that he will try other suboptimal actions or wait for a while to receive other advice from the coach. This, as one can imagine, will slow the rookie's learning process. On the contrary, if the rookie can memorize and reuse the previous advice, then the rookie's learning will be accelerated by simply adopting the optimal action advised by the coach previously.

To this end, this paper studies if and how a student (e.g., the rookie) benefits from reusing previous advice from a more experienced teacher (e.g., the coach). Building upon the existing teacher–student frameworks [2, 9], we propose a Memory-Based Teacher–Student Framework (MBTSF) such that agents are able to memorize and reuse previous advice. When a student receives advice in a state, the state is labeled as 'advised', and the state-advice pair $\langle state, advice \rangle$ is stored. At every time step, each agent can choose among learning by itself, asking for advice, and reusing the previous advice if available. As agents have no prior knowledge, their policies and their advice are generally not optimal at the early stage of learning. To avoid the use of non-optimal previous advice, which may hinder the student's performance, we propose two methods to decide whether previous advice should be reused. (i) Q-Change per Step (QChange): a student will reuse previous advice if this advice leads to an increase in the Q-value (i.e., the expected return). (ii) Decay Reusing Probability (Decay): a student will reuse previous advice with a decaying probability, since the previous advice may be outdated in a stochastic environment. In our experiments, we consider three different RL tasks: Mario, Predator–Prey and Half Field Offense. We show that in all these tasks, the reuse of previous advice significantly accelerates the students' learning.

The rest of this paper is organized as follows. Section 2 discusses the related works. Section 3 presents a brief background on RL and teacher–student frameworks. Section 4 describes the proposed framework MBTSF and two proposed methods (QChange and Decay) for deciding the reuse of previous advice. Section 5 provides the settings, results and analysis of our experiments. The last section concludes this paper with a discussion of future works.

2 Related works

Action advising problems [8, 9, 11, 25] concerns two roles for agents: the teacher and the student. A more experienced teacher agent helps accelerate the learning of a student agent by suggesting actions to be taken in certain states.

Early works [2, 7, 12, 24, 25, 29, 31] assume the roles of agents are fixed beforehand, and a human expert or a pre-trained agent takes the role of the teacher. Clouse [7] focuses on human teacher agents and proposes that a teacher should help a student agent whenever the student asks for advice. Torrey and Taylor [24, 25] propose a teacher–student framework, which allows a teacher to decide in which states to suggest actions to a student. They also introduce the concept of budget constraints that models limited communication between the teacher and student, in terms of the number of times that a teacher can provide action advice. Zimmer et al. [31] view teaching under the budget constraint to be a reinforcement learning problem and propose to learn when a teacher gives advice for a more efficient budget consumption. The teacher's reward decreases if the student spends more time steps in reaching its goal, which makes the teacher try to accelerate the student's learning through advising appropriately.

The jointly-initiated advising proposed by Amir et al. [2] assumes that an advisor-advisee relationship is built under the agreement of both the teacher and the student. A student can ask for advice if unsure what to do in a state, and a teacher who follows a fixed policy provides action advice if it considers the student's state is important to be advised. Fachantidis et al. [12] investigate how to select an appropriate teacher from several pre-trained agents to generate advice. They reveal that the agent who achieves the best average score when acting alone may not be the best teacher. Zhan et al. [29] consider the possibility of receiving advice from multiple teachers. They propose to use a majority vote to combine suboptimal advice to make the teacher–student framework more robust, but without considering the optimality of the teachers.

The aforementioned works focus on single-student action advising settings, and assume that the roles of teacher and student are predetermined in advance. More recently, Silva et al. [9] extend the teacher–student framework to a multiple teacher–student setting and propose an ad hoc advising framework. At every time step, each agent may take a student's role to ask for advice, while it can also take a teacher's role to advise other students. Being a student or a teacher depends on an agent's familiarity with the state to be advised, i.e., the number of times it has visited that state. Further, a teacher's policy is not necessarily fixed or optimal and can be improved through learning from other agents' advice. Wang et al. [26] incorporate a teacher–student framework to improve the speed of convention emergence in multi-agent systems. Different from the ad hoc advising framework, a student asks each of its neighbors in a network topology for advice, instead of broadcasting a single request to all of the other agents.

Ilhan et al. [13] combine the ad hoc advising framework with deep reinforcement learning to tackle continuous state space. As the state space is large, random network distillation is utilized to measure how frequently a state has been visited through non-linear function approximation. Then each Deep Q-learning agent can ask for advice if it has rarely visited a state, while it can give advice if it has visited a state frequently. Silva et al. [10] consider that counting the number of visits ignores the uncertainty of an agent. They utilize a multi-head version of Deep Q-network, where each head predicts a different estimate of the Q-values based on different samples of experience. By updating the network weights, the variance of the predictions is gradually reduced and used to estimate the uncertainty for each state. Agents with high uncertainty will decide to ask for advice, while those with low uncertainty will decide to provide advice.

There exist scenarios where agents may need to coordinate their behaviors. For example, when they have to go in opposite directions with similar positions, using the best actions from the teacher's view may lead to suboptimal joint return. Omidshafiei et al. [19] propose Learning to Coordinate and Teach Reinforcement to allow two agents to decide when and which action to be advised. Each agent learns to interact with the environment while learning when to take the role of teacher or student with a high-level reinforcement learning. In a student's role, an agent learns to choose whether to ask for advice or not. In a teacher's role, an agent learns to choose whether to give advice or not and decides to what to advice: either a piece of empty advice or an action from the student's action space. On top, a centralized Actor-Critic algorithm is employed for coordinating student and teacher activities as a high-level problem.

Notably, prior works primarily deal with the following three subproblems: (1) when a student asks for help; (2) when a teacher advises; (3) which particular action a teacher should advise. However, students generally discard the actions advised by teachers right after use. In this paper, we conjecture that reusing prior teacher advice can save the communication budget for a student and improve its learning and propose two different ways for the efficient reuse of prior advice. Our extensive experiments confirm that learning by reusing previous advice consumes fewer budgets and significantly improves the learning speed for students.

This paper is an extension of our earlier version [30]. In [30], we propose several advice reusing methods for students to decide how to reuse previous advice from teachers. In this version, we unify those methods and propose a new framework, Memory-Based Teacher–Student Framework (MBTSF), to highlight the usage of an extra memory space for students to store and reuse advice when necessary. Moreover, during implementation, we combine our proposed advice reusing methods with another popular advising method, *Ask Uncertainty-Importance Advising* (AUIA) [2], to illustrate that our proposed MBTSF can be applied to different advising mechanisms. In particular, our experimental results confirm the significant improvement induced by our advice reusing methods over the original AUIA.

3 Preliminaries

This section summarizes our selection of relevant background knowledge and techniques from recent research on Action Advising required for presenting our approach.

3.1 Single-agent RL and MARL

Reinforcement Learning (RL) enables the solution of Markov Decision Processes (MDPs) [23]. An MDP is described by a tuple $\langle S, A, T, R, \gamma \rangle$, where S is a set of environment states, A is a set of actions an agent can take, $T : S \times A \times S \rightarrow [0, 1]$ are transition probabilities between states, $R : S \times A \rightarrow \mathbb{R}$ is a reward signal, and $\gamma \in [0, 1)$ is a discount factor. At each time step, an RL agent observes the environment's state $s \in S$ and selects an action $a \in A$ to execute. Then the agent receives a numerical reward r from the environment and observes the next state $s' \in S$. The goal of an agent is to learn a policy $\pi : S \rightarrow A$ that maps states to actions such that the policy will maximize the expected cumulative discounted reward.

Temporal difference (TD) RL algorithms, such as Q-learning [28] and SARSA [23], learn a Q-value $Q(s, a)$ for each state-action pair (s, a), which estimates the expected return of an agent if it takes action a in state s. The Q-value $Q(s, a)$ is updated based on the following rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \times \delta \tag{1}$$

where $\alpha \in [0, 1]$ is the learning rate, and δ is the TD error. In Q-learning, we have:

$$\delta = r + \gamma \max_a Q(s', a) - Q(s, a) \tag{2}$$

where $\gamma \in [0, 1)$ is the discount factor, and s' is the next state after taking action a in the current state s. In SARSA, we have:

$$\delta = r + \gamma Q(s', a') - Q(s, a) \tag{3}$$

where a' is the action that the agent will execute in the next state s' according to its policy. A common policy for balancing exploration and exploitation is ϵ-*greedy* [23]. With a small probability ϵ an agent takes a random action, while with a large probability $(1 - \epsilon)$ it takes the action with the highest Q-value.

Q(λ) [23] and SARSA(λ) [20] are the extensions of Q-learning and SARSA, respectively. At every time step, instead of updating the Q-value for one single state, Q(λ) and SARSA(λ) improve the learning speed by using the TD error of the current time step to update the Q-values from past states within the training episode. $\lambda \in [0, 1]$ is a trace decay factor that controls the impact of TD errors on the Q-values from past time steps. As the number of past Q-value updates increases, current TD error has a smaller influence on these Q-values.

In the multi-agent case, we are interested in cooperative RL agents getting local observations and learning in a decentralized fashion [17]. They jointly affect the environment and receive the same reward while learning individual policies without accessing knowledge beyond the environment. The learning problem of multiple decentralized agents with local observations is generally modelled as a Decentralised partially observable Markov decision process (Dec-POMDP) [18], which is an extension of the Markov Decision Process (MDP)

[23]. A Dec-POMDP is defined by a tuple $\langle \mathcal{I}, \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O}, \gamma \rangle$, where $\mathcal{I}$ is the set of n agents, $\mathcal{S}$ is the set of environment states, $\boldsymbol{\mathcal{A}} = \times_{i \in \mathcal{I}} \mathcal{A}_i$ is the set of joint actions, $\mathcal{T}$ is the state transition probabilities, $\mathcal{R}$ is the reward function, $\boldsymbol{\Omega} = \times_{i \in \mathcal{I}} \Omega_i$ is the set of joint observations, $\mathcal{O}$ is the set of conditional observation probabilities, and $\gamma \in [0, 1)$ is the discount factor. At every time step t in an environment state s', each agent i perceives its individual observation o_t^i from one joint observation $\mathbf{o} = \langle o^1, ..., o^n \rangle$ determined by $\mathcal{O}(\mathbf{o}|s', \mathbf{a})$, where $\mathbf{a} = \langle a^1, ..., a^n \rangle$ is the joint action that causes the state transition from s to s' according to $\mathcal{T}(s'|\mathbf{a}, s)$, and receives the same reward r determined by $\mathcal{R}(s, \mathbf{a})$.

3.2 Teacher–student framework

Here we introduce two typical teacher–student frameworks for determining when to ask for advice and give advice, namely, *Ask Uncertainty-Importance Advising* [2] and *AdhocTD* [9]. In these two frameworks, the number of times that a student can ask for advice and that a teacher can give advice is constrained by two numeric budgets b_{ask} and b_{give}, respectively. The algorithms continue until the inquiry budget finishes.

3.2.1 Ask Uncertainty-Importance Advising (AUIA)

This framework is an instance of the jointly-initiated advising framework [2]. A learning student agent and a trained teacher agent use respectively, *Ask Uncertainty* and *Importance Advising*, to decide advising opportunities. The Importance Advising method produces advice by querying the teacher's learned value function for every state the student faces so that the budget is better spent on the states considered to be more important. The importance of a state s is computed as follows:

$$I(s) = \max_a Q^{teacher}(s, a) - \min_a Q^{teacher}(s, a) \tag{4}$$

where the state importance $I(s)$ encodes the difference between the best and the worst actions. A teacher will give advice in state s if $I(s)$ exceeds a threshold t_g. In AUIA, *Ask Uncertainty* enables a student to ask for advice when it is uncertain about which action to take in state s, such that

$$\max_a Q^{student}(s, a) - \min_a Q^{student}(s, a) < t_a \tag{5}$$

where t_a is the student's threshold for uncertainty.

3.2.2 AdhocTD

AdhocTD is a multi-agent teacher–student framework with no pre-trained agent taking the role of teacher, and all of the agents advise one another to accelerate learning. In a state s, an agent asks for advice with an inquiry probability P_{ask} calculated as follows:

$$P_{ask}(s) = (1 + v_a)^{-\sqrt{n_{visit}(s)}} \tag{6}$$

where v_a is a predetermined parameter, $n_{visit}(s)$ is the number of times that the agent has visited state s. If the agent visits state s very few times, the value of $n_{visit}(s)$ is low, resulting in a higher probability $P_{ask}(s)$ of asking for advice. Each of the other agents advises the action having the highest Q-value in state s with a giving probability $P_{give}(s)$ calculated as follows:

$$P_{give}(s) = 1 - (1 + v_b)^{-\sqrt{n_{visit}(s)} \times I(s)} \tag{7}$$

where v_b is a predetermined parameter, and $I(s)$ is the state importance defined in Eq. 4. When another agent visits state s many times or the importance of state s is high, the other agent provides its advice with a higher probability. If a student is advised by more than one teacher, it uses the action advised by most of the teachers.

4 Proposed methods

In this research, we are interested in investigating how a student agent's learning benefits from reusing previous advice. Our proposed framework Memory-Based Teacher–Student Framework (MBTSF), enables each agent to memorize advice from their teachers for necessary reuse. Since a teacher agent may still be adjusting its policy when it gives advice, the advised actions are likely to be non-optimal, and thus exploration needs to be allowed for a student when it reuses previous advice. In MBTSF, each agent uses a memory space to store advice from others and decides how long the advice should be memorized. Every time a student agent encounters a state advised before, it chooses between (1) reusing previous advice, (2) asking for new advice, and (3) learning by following a usual exploration strategy.

Next, we start by providing an overview of MBTSF in Sect. 4.1. Then, we describe the core components of MBTSF, which are how a teacher generates advice and how a student reuses previous advice stored in its memory space, respectively in Sects. 4.2 and 4.3.

4.1 Memory-based teacher–student framework

In the proposed framework, each agent can take the role of student, teacher, or both of them simultaneously. So, they may ask for advice (i.e., being a student), but provide advice for others (i.e., being a teacher) at the same time step. Based on student's role and teacher's role, an agent's action selection process comprises the following procedures: first, *advice reusing*, where the agent checks whether previous advice is available in the current state through memory space; next, *advice generation*, where the agent asks for a piece of new advice if no previous advice is reused; finally, *exploration by itself*, where the agent uses its intended action. The process of advice reusing and advice generation continues until the budget runs out.

4.1.1 Student's role

A student agent has no prior knowledge about the task it performs. The student agent, who indeed receives payoffs from the environment and learns by exploration, can also accelerate learning by taking action suggestions from teachers in certain states. Furthermore, the student is allocated with a memory space to memorize the advice that has been actually executed in the task.

During the learning process, each agent decides when to ask for advice and how to use advice according to the student's role $R_{student}$. We define student role $R^i_{student}$ for an agent i as a tuple $\langle P^i_{ask}, b^i_{ask}, G^i, \Gamma^i, \mathcal{M}\rangle$:

- $P^i_{ask} : S \rightarrow [0, 1]$ is the probability function for agent i to become a student. The function maps every state to an inquiry probability. In current state s, before choosing which action to take, agent i sends a request with a probability $P^i_{ask}(s)$ to each of the other agents for asking action advice. Some traditional teacher–student frameworks, for example, Ask Uncertainty-Importance Advising, using deterministic asking function, can be transformed to function P^i_{ask}, in which agents choose 'yes' to seek advice and 'no' to learn individually. In this case, P^i_{ask} can be defined as follows,

$$P^i_{ask}(s) = \begin{cases} 1, & \text{if the student decides to ask for advice} \\ 0, & \text{otherwise} \end{cases}. \tag{8}$$

- b^i_{ask} is the inquiry budget for agent i to ask for advice. At the current time step, b^i_{ask} decreases by 1 if the student is advised (even by more than one teacher), which implies a valid communication. When the budget runs out, the student stops both asking for advice and reusing previous advice, and learns individually at the following time steps.
- G^i is a collection of the other agents that can receive the request of agent i (i.e., the student). In this paper, we define that G^i consists of all of the other agents whose budget of giving advice does not run out.
- Γ^i is a function that combines all received advice if multiple teachers provide advice to student i, and finally, an action will be selected among available advice for execution in the environment. A general implementation is a majority vote, which chooses the advice suggested by most teachers.
- $\mathcal{M} = \{..., \langle s_t, a_t\rangle, ...\}$ is a set of state-advice tuples which are selected and executed in the environment by student i. $\mathcal{M}$ requires extra memory space. However, the maximum capacity of memory $\mathcal{M}$ is bounded by the inquiry budget b^i_{ask} as consuming one budget means at most one advice will be performed and memorized finally. Every update of memory M provides student i a chance to keep track of teachers' current learning.

4.1.2 Teacher's role

A teacher is a role played by a human expert or an artificial agent. The teacher either adopts a pre-trained policy to advise student's learning, or follows an non-optimal policy and still needs to improve itself.

The most significant aspect of MBTSF is that each agent stores advised actions from teachers and reuse them if needed. Note that each agent may take the roles of student and teacher simultaneously if the agent encounters a (local) state that lacks of learning, or has already acquired more experience in other regions of the state space. Now we consider how another agent j takes the role of teacher and gives its advice. We define the teacher's role $R^j_{teacher}$ for agent j as a tuple $\langle P^j_{give}, b^j_{give}, \phi^j\rangle$.

- $P^j_{give} : S \rightarrow [0, 1]$ is the probability function of agent j that maps a student's current state to an advising probability to response the student's inquiry. If agent j receives multiple requests, the current states of each student are fed into function P^j_{give} to output a vector of advising probabilities, then processing corresponding requests simultaneously. Similar to asking function, deterministic advising function used by previous methods (e.g., Ask Uncertainty-Importance Advising) can be absorbed into function P^j_{give} by using the following definitions,

$$P^j_{give}(s) = \begin{cases} 1, & \text{if agent} j \text{ decides to give advice,} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

– b^j_{give} is the giving budget of agent j to share its advice. b^j_{give} is decremented by 1 every time the teacher offers his advice to a student. Once the budget is used up, agent j will not provide advice anymore.
– ϕ^j is an operator that maps an agent's policy π_j to advice.

$$\phi^j : \pi_j \to A \tag{10}$$

In general, teachers and students share the same action space A, and the student can directly perform the advised action in the environment. In teacher–student frameworks, as agents involved advising relationships are assumed to have the same optimal policy, a teacher usually uses the action with the highest Q-value as advice to the associated students. We follow the same setting in this paper, and therefore operator ϕ^j can be viewed as a maximum operation, which selects the action corresponding to teacher agent j's maximum Q-value for the requested state.

4.2 Advice generation

We assume that agent i arrives at an unknown state s_t (for time step t) so that no previous advice is available for the current learning. Agent i needs to explore how to take the role of student, as described in Algorithm 1. In the current state s_t, agent i firstly checks if the inquiry budget b^i_{ask} is still available (Line 2). With the asking probability calculated by function P^i_{ask}, student i initiates an inquiry and broadcasts to potential teachers from collection G^i, e.g., all other agents in the environment (Lines 3–4). The request is either composed of the student's current state to let a teacher know in which state they should provide advice, or simply denoted as a requesting signal if agents in G^i are able to recognize the observation of student i. Then student i pauses its learning and waits for a predetermined time interval to collect the answer. Nevertheless, in the current time step, it still has a chance to take a teacher's role for any other requested states.

After receiving the request from student i, another agent j (from collection G^i) suspends its current learning and determines how to give advice, as described in Algorithm 2. Agent j firstly switches its own state to the requested state s_t. If the current giving budget b^j_{give} is greater than 0, agent j will evaluate the possibility of taking a teacher's role through function P^j_{give} (Lines 2–3). We assume that all agents are equipped with similar state representation. Thus, advice can be easily derived by selecting the action with the highest Q-value in state s_t (line 4). Every time advice is given, the giving budget b^j_{give} is decremented by 1 (Line 5). Then, teacher j sends back its advice to associated students and switches back to its original state, unless there are other requests that have to be processed.

The communication channel between student i and teachers will be closed in state s_t once the time interval has been reached. Note that if no advice is shared or no previous advice is available, the student learns individually as usual. After receiving all advice, student i feeds those advised actions into combination function Γ^i to select the action that will be executed in state s_t, as shown in Algorithm 1. The inquiry budget b^j_{ask} is also decremented by 1 if student i does receive action advice (Line 11). Then the Q-value of that particular state-action pair is updated by incorporating environmental rewards. Now, the remaining part of MBTSF is to design a proper way to effectively use previous advice when student i visits the same state s_t in the future.

```
Algorithm 1 Advice Generation for Student $i$
Require: function $P^i_{ask}$, inquiry budget $b^j_{give}$, collection $G^i$, combination function $\Gamma^i$
 1: let current state be as $s_t$
 2: if $b^i_{ask} > 0$ then
 3:     if $P^i_{ask}(s_t) > getRandomValue(0,1)$ then
 4:         Get a set of available agents $G^i(s_t)$
 5:         Define $\Pi$ as the set of received advice
 6:         for each agent $j$ from $G^i(s_t)$ do
 7:             $\Pi \leftarrow \Pi \cup j.GivingAdvice()$
 8:         end for
 9:         if $\Pi \neq \emptyset$ then
10:             $a_t \leftarrow \Gamma^i(\Pi)$ (e.g., majority vote)
11:             $b^i_{ask} \leftarrow b^i_{ask} - 1$
12:             return selected advice $a_t$
13:         end if
14:     end if
15: end if
16: return $\emptyset$
```

```
Algorithm 2 Giving Advice by Teacher $j$
Require: giving function $P^j_{ask}$, giving budget $b^j_{give}$
 1: Switch from its own state to requested state $s_t$
 2: if $b^j_{give} > 0$ then
 3:     if $P^j_{give}(s_t) > getRandomValue(0,1)$ then
 4:         $a \leftarrow \arg\max_a Q^j(s_t, a)$
 5:         $b^j_{give} \leftarrow b^j_{give} - 1$
 6:         return advice $a$
 7:     end if
 8: end if
 9: Switch back to its own state
10: return $\emptyset$
```

4.3 Learning by reusing previous advice

Suppose that all advised state-action pairs are recorded to memory $\mathcal{M}$ forever. By reusing advice from memory, however, the student is imitating the teacher's behaviors rather than learning a policy. On the other hand, when all agents are learning from scratch, their policies are most likely to be non-optimal. Allowing a student to explore the environment occasionally provides the student a chance to surpass the teacher. Moreover, with increased training time, previous advice might be outdated. Therefore, new advice must be added to memory timely so that a student can follow teacher's latest advice. Also, learning by reusing previous advice needs to consider when to reuse advice and how to update advice memory, in order to balance the trade-off between insisting on previous advice, asking for new advice, and using currently learned policy.

4.3.1 Q-Change per step

As student agents aim to learn a policy which can bring long-term rewards, intuitively, advice is beneficial if it leads to an increase of estimated cumulative rewards, i.e., the Q-values. On the other hand, teachers advise actions that have the highest Q-values in requested states. Therefore, the student's Q-value corresponding to an advised action should surpass the Q-values of the other actions. Inspired by this, we propose our first implementation of the advice reusing process in our MBTSF (memory-based teacher–student framework), named *Q-Change per Step*. In this method, a student agent will evaluate the difference in Q-values corresponding to advice before and after following that advice in the environment. If the difference is significant enough, for example, exceeding a threshold, then the advice will be stored and reused in future time steps.

Algorithm 3 Advice Reusing with Q-Change per Step

```
Require: agent $i$, memory $\mathcal{M}$, asking budget $b^j_{give}$, threshold $\eta$
 1: Initialize advice memory $\mathcal{M}$
 2: for every training step $t$ do
 3:     Observe current state $s_t$
 4:     Denote $a_t$ as the finally selected action in state $s_t$
 5:     if $b^i_{ask} > 0$ then
 6:         if $s_t \in \mathcal{M}$ then
 7:             Set $a_t$ as the corresponding advice in $\mathcal{M}$ indexed by $s_t$
 8:         end if
 9:     end if
10:     if no advice is reused then
11:         if $AdviceGeneration() \neq \emptyset$ then
12:             $a_t \leftarrow AdviceGeneration()$
13:             if $s_t$ exists in $\mathcal{M}$ then
14:                 Replace by the new advice pair $\langle s_t, a_t \rangle$
15:             else
16:                 Store advice pair $\langle s_t, a_t \rangle$ in $\mathcal{M}$
17:             end if
18:         end if
19:     end if
20:     if no advice is reused or suggested then
21:         Set $a_t$ as the action selected from usual exploration strategy (e.g., $\epsilon$-greedy)
22:     end if
23:     Perform action $a_t$ in the environment
24:     if an advice is executed then
25:         if $Q^i_{t+1}(s_t, a_t) - Q^i_t(s_t, a_t) \geq \eta$ then
26:             Keep advice pair $\langle s_t, a_t \rangle$ in $\mathcal{M}$
27:         else
28:             Remove advice pair $\langle s_t, a_t \rangle$ from $\mathcal{M}$
29:         end if
30:     end if
31: end for
```

Algorithm 3 describes *Q-Change per Step* from the perspective of student agent *i*. At time step *t*, the agent firstly gives priority to checking whether previous advice is available for the current state s_t (with a subscript time step t) (Lines 1–4). When budget b_{ask} is exhausted prematurely in the early stage of learning, the reusing process will be terminated for avoiding sub-optimal suggestions having too much influence on students' learning (Line 5). When the budget b^i_{ask} is larger than 0, agent i checks whether state s_t is recorded in memory $\mathcal{M}$. If yes, the stored advice is picked up and used as the finally executed action (Lines 6–8). However, once the agent decides not to reuse previous advice, advice generation will be executed. When agent i's request is responded, new advice chosen by the agent is viewed as the action to be performed in state s_t. At the same time, the advised action and state s_t are stored in memory $\mathcal{M}$ for reuse next time. Specifically, if state s_t is advised before, the original advised action in memory $\mathcal{M}$ will be replaced by the latest one aligning with the teachers' latest knowledge (Lines 10–19). As long as no advice is reused or suggested, agent i will learn by itself immediately, i.e., selecting an action from the usual exploration strategy like ϵ-greedy (Lines 20–22). In *Q-Change per Step*, agents will record their Q-values before and after executing the advice in the environment, in order to evaluate the impact of the advice on agent i's learning process. In state s_t, if the change of agent i's Q-value corresponding to the advice exceeds (or equals) a predefined threshold η, then the agent keeps the advice in advice memory $\mathcal{M}$, otherwise dismisses it (Lines 24–30).

4.3.2 Decay reusing probability

In many scenarios, agents face a stochastic and noisy environment, where the variance of Q-values could mislead the decision of reusing previous advice in *Q-Change per Step*. In those situations, agents using *Q-change per Step* may have less chance to reuse previous advice, reducing the impact of teachers on students' learning. In the meantime, students also need to avoid the impact of early versions of teachers, to have a chance to explore the environment and evaluate their own policies. Motivated by this, we propose a more flexible and robust implementation of the advice reusing process, named *Decay Reusing Probability*, such that agents use decaying probability (starting from 1) to reuse advice without abrupt termination as in *Q-change per Step*, and an increasing probability (starting from 0) to ask for new advice and explore their own strategies.

Consider one of the most popular exploration strategies, the $\epsilon-$greedy exploration. Agents have four ways to select an action in the environment: reusing previous advice, asking for new advice, exploring the environment, and exploiting its own policy. In current state s_t, agent i chooses an action a_t to perform in the environment according to the following probabilities,

$$a_t = \begin{cases} \text{Previous Advice} & \text{w.p. } P^i_{reuse} \\ \text{Asking for Advice} & \text{w.p. } (1 - P^i_{reuse}) \times P^i_{advice} \\ \text{Greedy Action} & \text{w.p. } (1 - P^i_{reuse}) \times (1 - P^i_{advice}) \times (1 - \epsilon) \\ \text{Random Action} & \text{w.p. } (1 - P^i_{reuse}) \times (1 - P^i_{advice}) \times \epsilon \end{cases} \tag{11}$$

where P^i_{reuse} is the probability for agent i to reuse previous advice, and P^i_{advice} is the probability for agent i to obtain new advice from other agents. In MBTSF, P^i_{advice} is defined as follows:

$$P^i_{advice} = P^i_{ask} \times \max_{j \in G^i} P^j_{give} \tag{12}$$

P^i_{advice} summarizes the maximum probability that agent i's inquiry request is responded by teachers (i.e., agents from collection G^i), and is determined jointly by asking function P^i_{ask} and giving function P^j_{give}. Reusing probability P^i_{reuse} represents whether student agent i

should reuse teachers' advice to guide its action selection. Hence, agent i follows previous advice with probability P^i_{reuse}, asks for a new advice with probability $(1 - P^i_{reuse}) \times P^i_{advice}$, exploits its Q-values with probability $(1 - P^i_{reuse}) \times (1 - P^i_{advice}) \times (1 - \epsilon)$, and acts randomly with remaining probability, where the total probability of all choices equals to 1.

Now we propose a way to construct reusing probability P^i_{reuse}. Intuitively, in the early stage of the reusing process, a student probably expects to have more opportunities to reuse existing advice since it has no better options. As the number of times of advice being reused increases, the student would prefer to get rid of old advice and follow the teacher's latest learning. Then we define P^i_{reuse} for agent i in the current state s_t as follows,

$$P^i_{reuse}(s_t) = \rho^{m_{visit}(s_t)} \tag{13}$$

where decay value $\rho \in [0, 1]$ is a hyperparameter and m_{visit} is the number of times that the current advice is performed in state s_t. Whenever agent i chooses to ask for help and receives a new advice in state s_t, m_{visit} is reinitialized as 0. Then, the value of reusing probability for that advice will be 1, indicating that agent i will try the advice at least once more to enhance the impact of teachers on the student. Counter m_{visit} is accumulated and refreshed along with learning to ensure that agent i can follow the teacher's latest learning. If decay parameter $\rho \in (0, 1)$, reusing probability $P^i_{reuse}(s_t)$ decreases exponentially as agent i repeatedly performs the latest advice in state s_t. When the decay parameter ρ is 0, agent i is not able to reuse advice, which is equal to learning in the traditional teacher–student framework. When the decay value ρ is 1, agent i simply follows the teacher's advice rather than learning a policy.

Algorithm 4 Advice Reusing with Decay Reusing Probability

```
Require: agent $i$, memory space $\mathcal{M}$, asking budget $b^j_{give}$, reusing probability $P^i_{reuse}$, decay
    value $\rho$
 1: Initialize advice memory $\mathcal{M}$
 2: for every training step $t$ do
 3:     Observe current state $s_t$
 4:     Denote $a_t$ as the finally selected action in state $s_t$
 5:     if $b^i_{ask} > 0$ then
 6:         if $P^i_{reuse}(s_t) > getRandomValue(0, 1)$ then
 7:             Set $a_t$ as the corresponding action in $\mathcal{M}$ indexed by $s_t$
 8:             $P^i_{reuse}(s_t) \leftarrow P^i_{reuse}(s_t) \times \rho$
 9:         end if
10:     end if
11:     if no advice is reused then
12:         if $AdviceGeneration() \neq \emptyset$ then
13:             $a_t \leftarrow AdviceGeneration()$
14:             if $s_t$ exists in $\mathcal{M}$ then
15:                 Replace by the new advice pair $\langle s_t, a_t \rangle$
16:             else
17:                 Store advice pair $\langle s_t, a_t \rangle$ in $\mathcal{M}$
18:             end if
19:             $P^i_{reuse}(s_t) \leftarrow 1$
20:         end if
21:     end if
22:     if no advice is reused or suggested then
23:         Set $a_t$ as the action selected from usual exploration strategy (e.g., $\epsilon$-greedy)
24:     end if
25:     Perform action $a_t$ in the environment
26: end for
```

Algorithm 4 describes advice reusing process with *Decay Reusing Probability*. At time step t, agent i observes the current state s_t (Lines 2–3). Firstly, the agent decides whether to reuse previous advice from its memory. Then the probability of reusing advice is decayed by multiplying decay parameter ρ, leading to a lower chance to choose previous advice in state s_t (Lines 6–9). Then, if no advice is reused, agent i decides to ask for new advice through advice generation. If state s_t exists in memory $\mathcal{M}$, previous advice will be replaced by the latest one to follow the teacher's learning. The associated reusing probability for state s_t is then initialized with 1 (Lines 11–21). Finally, for the situation where no previous advice is reused or no new advice is suggested, agent i performs its intended action selected by ϵ−greedy (Lines 22–24).

4.3.3 Discussion

Q-Change per Step and *Decay Reusing Probability* are two implementations of advice reusing in MBTSF. The former utilizes Q-values to decide whether the advice is beneficial for obtaining more rewards. Specifically, threshold η evaluates the minimum increase in the Q-values for an advised action to be reused. Due to the variance in Q-value estimation, the amount of increase in Q-values may fluctuate over time, which prevents students from constantly reusing teachers' advice. Thus, *Q-Change per Step* would be more effective in domains where the advice can be misleading or of low quality. In this case, students can easily get rid of the negative impact of teachers. By contrast, *Decay Reusing Probability* provides a more stable way to reuse teachers' advice, by incorporating a decaying probability for deciding whether previous advice should be reused before asking a new one. If the same advice is likely to remain useful for a while, *Decay Reusing Probability* should be better.

5 Experiments

Traditional teacher–student frameworks can be used to generate advice for MBTSF. To demonstrate the flexibility of our proposed methods, we combine the two implementations of advice reusing process in MBTSF, *Q-Change per Step* and *Decay Reusing Probability*, with two popular advising methods, *Ask Uncertainty-Importance Advising* [2] and *AdhocTD* [9]. Then previous advising methods are embedded into our framework to decide when to ask for advice and give advice. In the experiments, we compare the performance of the following methods,

(a) **Independent Learners (IL)** [6]. Each agent learns independently without communication. IL is served as a baseline method to validate the benefit of advising and reusing advice in different tasks.
(b) **Ask Uncertainty-Importance Advising (AUIA)** [2]. Importance-based methods rely on the range of Q-values for a certain state, which is a widely used metric for determining when to give advice. There are various forms of Importance-based advising frameworks. In order to balance budget constraints and experimental results, we use the version of AUIA.
(c) **AUIA-QChange/Decay.** AUIA can be combined with our proposed advice reusing methods *Q-Change per Step* (i.e., AUIA-QChange) and *Decay Reusing Probability* (i.e., AUIA-Decay) to generate advice. Therefore, Eqs. 5 and 4 are used in AUIA-QChange/Decay to decide when to ask for and give advice. Then, the asking function P_{ask} and the giving function P_{give} output probability 1 for asking and giving advice when a state is uncertain for a student, i.e., $\max_a Q^{student}(s,a) - \min_a Q^{student}(s,a) < t_a$ and important for a teacher, i.e., $\max_a Q^{teacher}(s,a) - \min_a Q^{teacher}(s,a) > t_g$. Otherwise, agents will not ask for advice or give advice, i.e., $P_{ask} = P_{give} = 0$. AUIA, AUIA-QChange and AUIA-Decay share the same parameters t_a and t_g. We fine-tune the values of t_a and t_g based on empirical experience and hence their values are generally different across different scenarios.
(d) **AdhocTD** [9]. AdhocTD is the state-of-the-art multi-agent advising approach that has been shown to largely reduce advising budget and achieve excellent results in a soccer game.
(e) **AdhocTD-QChange/Decay.** AdhocTD can be viewed as an ideal advice generation algorithm. We combine AdhocTD with our proposed methods *Q-Change per Step* (QChange) and *Decay Reusing Probability* (Decay), respectively. Therefore, the asking function P_{ask} (i.e., Eq. 6) and the giving function P_{give} (i.e., Eq. 7) proposed in AdhocTD are directly used in AdhocTD-QChange/Decay to decide when to ask for advice and give advice. AdhocTD, AdhocTD-QChange and AdhocTD-Decay share the same parameters v_a and v_b, which will be chosen empirically or fine-tuned for the following experiments.

We conduct experiments on three typical reinforcement learning tasks: Mario, Predator–Prey, and Half Field Offense. Mario is a complex stochastic game in which a single agent, “Super Mario”, learns to obtain higher scores. In Mario, the teacher’s role is taken by another agent who has already learned a fixed policy. For AUIA and AdhocTD and corresponding advice reusing methods, we record the number of times that the teacher visits each state and its Q-values after following a fixed policy. Predator–Prey is a popular benchmark for multi-agent learning. Half Field Offense is a difficult robot soccer game, where agents must take stochastic action effects and noisy observations into account. In both Predator–Prey and Half Field Offense, all agents learn from scratch without using predefined policies, and they accelerate joint learning by advising one another and reusing previous advice.

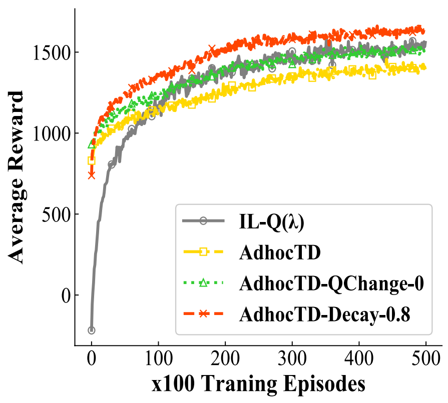
5.1 Mario

5.1.1 Game description

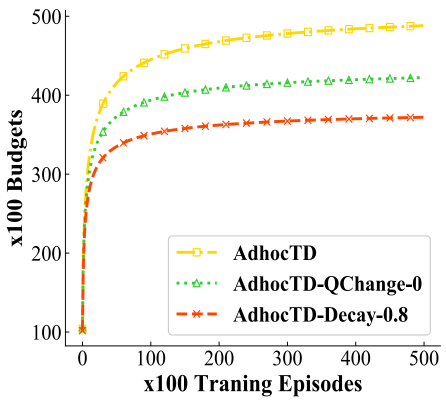
The Mario AI benchmark is based on Nintendo’s classic platform game Super Mario Bros. In this research, we use the publicly available code released by Karakovskiy and Togelius [14]. The Mario agent can walk and run to the right and left, jump, and shoot fireballs through 2-D levels, with the goal to collect as many points as possible. The state-space in Mario is fairly complex, as Mario observes all information pertaining to himself. We use 27 discrete state-variables, which are also adopted by previous works [22, 27]. The state features are composed of three boolean variables which define whether Mario is able to jump, shoot fireballs, or stand on a tile; two variables that define the direction of Mario’s movements on *x-dir* and *y-dir*; eight values that define whether there is an enemy in matching cell surrounding by the agent; another eight variables that keep track of the cells one step away from the immediate cells; four boolean variables that identify whether there are obstacles on the four vertical cells to the immediate right of Mario; and finally two variables that represent the Euclidean distance between the agent and the closest enemy on *x-dir* and *y-dir*. The agent can take 12 actions by combining one of each option from three sets: {left, right, no direction}, {jump, do not jump}, and {run, do not run}. Mario receives a reward of +10 for stepping on an enemy, +16 for collecting a coin, +24 for finding a hidden block, +58 for eating a mushroom, +64 for eating a fire-flower, +1024 for successfully finishing the level, -42 for getting hurt by an enemy (e.g., losing fire-flower), and -512 for dying. Mario game is designed as an episodic task. In every episode, the Mario agent plays a randomly generated level, starting from a randomly selected mode (small, large, and fire-Mario). The level is ended either with the agent’s success, the agent’s death, or a timeout of 200 seconds.

5.1.2 Parameters setup

We use *Average Reward per Episode* (ARE) to assess the Mario agent’s performance. ARE is the average reward per a certain amount of training episodes, and we set 100 to obtain a smooth learning curve. Higher ARE values mean better performance. As the whole training process contains 50,000 episodes, we can get 500 ARE values for a single run. Then we perform 60 runs for all methods to stabilize the performance. In Mario, we use $Q(\lambda)$ with a learning rate of $\alpha = 0.001$, a discount factor of $\gamma = 0.9$, and a decay rate of $\lambda = 0.5$ for all the experiments. Whenever an agent chooses the final actions to be performed, it will



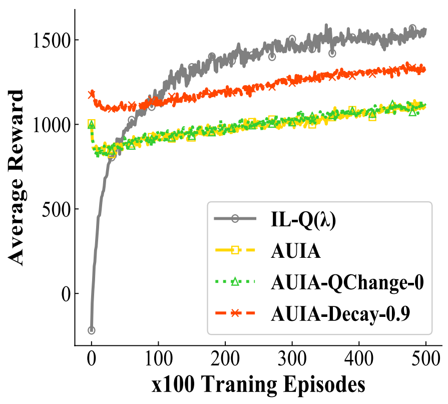
(a) Average reward per 100 episodes



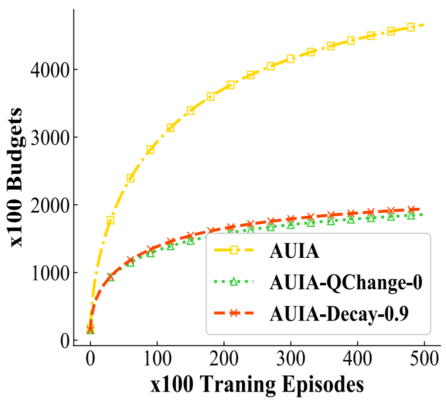
(b) The spent budget

Fig. 1 ARE and budget consumption of IL-Q(λ), AdhocTD and AdhocTD-QChange/Decay when $b_{ask} = b_{give} = 50,000$, $\eta = 0$, and $\rho = 0.8$

use $Q(\lambda)$ to update the corresponding Q-values. When no advice is reused or advised, the agents use ϵ-greedy as the exploration strategy with $\epsilon = 0.05$. The parameters of AdhocTD (i.e., v_a and v_b) and AUIA (i.e., t_a and t_g) can heavily affect the advising opportunities as well as corresponding action reusing methods. For example, when v_a is a small value and v_b is a high value, an agent in AdhocTD is more likely to ask for advice, and other agents are more likely to provide advice for him/her. Thus, the budget can be consumed quickly. On the contrary, with higher v_a and lower v_b, advice is suggested when a student visits the current state a few times while the teacher has experienced the student's state many times. The parameters for AdhocTD and AUIA are tuned respectively, to balance the budget consumption and performance. Then in AdhocTD, we choose $v_a = 2$ and $v_b = 0.2$, and the same values in AdhocTD-QChange/Decay. For AUIA, we use $t_a = 0.01$ and $t_g = 0.03$, and also the same values in AUIA-QChange/Decay. Then for AdhocTD-QChange/Decay



(a) Average reward per 100 episodes



(b) The spent budget

Fig. 2 ARE and budget consumption of IL-Q(λ), AUIA and AUIA-QChange/Decay when $b_{ask} = b_{give} = 500,000$, $\eta = 0$, and $\rho = 0.9$

Table 1 Performance metrics for the agents in Mario (average of over 60 trials)

Agents	Initial	Last	Mean	Budget
IL-Q(λ)	-218.061	1560.148	1318.743	-
AdhocTD	830.007	1393.961	1263.297	48,812
AdhocTD-QChange ($\eta = 0$)	931.293	1533.551	1370.024	42,233
AdhocTD-QChange ($\eta = 0.01$)	868.465	1408.128	1262.018	48,628
AdhocTD-QChange ($\eta = 0.03$)	844.979	1396.403	1258.330	48,764
AdhocTD-Decay ($\rho = 0.8$)	738.327	**1628.328**	**1483.522**	37,206
AdhocTD-Decay ($\rho = 0.9$)	847.532	1592.780	1453.173	36,120
AdhocTD-Decay ($\rho = 0.99$)	855.415	1489.469	1338.959	**34,219**
AUIA	1005.591	1105.782	989.924	465,944
AUIA-QChange ($\eta = 0$)	998.733	1110.647	997.067	185,631
AUIA-QChange ($\eta = 0.01$)	1024.009	1087.953	992.455	466,739
AUIA-QChange ($\eta = 0.03$)	967.605	1097.942	978.921	466,778
AUIA-Decay ($\rho = 0.7$)	1171.516	1209.140	1087.889	250,481
AUIA-Decay ($\rho = 0.8$)	**1198.386**	1225.343	1131.825	227,506
AUIA-Decay ($\rho = 0.9$)	1173.953	1315.205	1218.803	193,977

Higher values of Initial, Last and Mean are better, while lower values of Budget are better. The best value of each column is shown in bold

and AUIA-QChange/Decay, we investigate the effect of different values of threshold η and decay value ρ on the learning performance. Therefore, we choose: $\eta = 0$, 0.01, and 0.03 for both AdhocTD-QChange and AUIA-QChange; $\rho = 0.8, 0.9$, and 0.99 for AdhocTD-Decay, and $\rho = 0.7, 0.8$, and 0.9 for AUIA-Decay. For the budget setting, we use a sufficiently large budget to enable agents to ask for advice.

5.1.3 Performance comparison

Figure 1 shows the *Average Reward per Episode* (ARE) metric and consumed budget for IL-Q(λ), AdhocTD, and the corresponding action reusing algorithms with selected threshold η and decay value ρ in Mario. The results show that AdhocTD and our proposed methods benefit from asking (or reusing) advice in the beginning. Notably, AdhocTD-Decay with $\rho = 0.8$ performs much better than other methods and spends much less budget. Traditional advising framework, AdhocTD, consumes the largest budget, while achieves even lower ARE values than independent learners. In AdhocTD, agents ask for too much advice during learning, and they almost have no time to explore the environment. Due to reusing previous advice, AdhocTD-QChange has higher ARE values than AdhocTD. Even though AdhocTD-QChange surpasses IL-Q(λ) in the beginning, they end up with similar performance. This may be attributed to the fact that the teacher's suggestions in AdhocTD-QChange cannot have a long-term impact on student learning.

Figure 2 shows the ARE values and budget consumption achieved by independent learners, AUIA, and our proposed methods throughout the overall evaluation. Our experiment show that AUIA and corresponding action reusing methods (i.e., AUIA-Decay/QChange) have much lower ARE values than IL-Q(λ). Nevertheless, AUIA-Decay still obtains much higher ARE than AUIA and AUIA-QChange.

Table 1 summarizes the first ARE value (Initial), the last ARE value (Last), the average ARE of the whole training episodes (Mean), and the final consumed budget (Budget) of all methods. Notably, AUIA and corresponding action reusing methods spend significantly larger budget than other methods. At the beginning of training, since agents rarely explore the environment, consuming budget can quickly get more guidance from the teacher to improve agents' learning. However, sacrificing the opportunity to explore the environment eventually leads to worse performance than independent learning. In AdhocTD-Decay, with increased decay value ρ, agents have fewer opportunities to learn from the environment, resulting in lower Mean value. However, AUIA-Decay achieves better performance with higher decay value ρ. Another interesting finding in the table is that both AdhocTD-QChange and AUIA-QChange achieve the best results with the threshold $\eta = 0$. Higher values of η, like 0.01 or 0.03, make it more difficult for the agent to reuse previous suggestions. This inspires us that both inefficient usage of previous advice and asking for too much advice can hinder the student's learning.

5.2 Predator–Prey

5.2.1 Game description

Predator–Prey (PP) is a grid-based game that becomes a benchmark environment for evaluating multi-agent systems (MAS) before applying them in more complex situations. We use the publicly available instantiation of the PP domain [4]. In our experiments, four predators explicitly coordinate their actions to capture one prey in a discrete $N \times N$ grid environment, where N is the number of cells in each direction, and we set N =10. One cell is allowed to be occupied by only one agent to avoid deadlock. At each time step, four predators and the prey execute one of the five possible actions: *Stay*, *Go Up*, *Go Down*, *Go Left*, and *Go Right*. The predators are reinforcement learning agents and learn to catch the prey as soon as possible. To make the learning task more realistic, the prey moves according to a randomized policy: it takes a random action with a probability of 0.2, with the rest of the time moving away from all predators. By executing an action, each agent moves cell by cell in the corresponding direction. The prey is captured when four predators are located in cells adjacent to the prey in four cardinal directions separately. After a capture, predators and prey are set up in distinct random positions. One episode starts when four predators and the prey are initialized with random positions in the grid world. The episode ends either when predators catch the prey, or a time limit of 2500 is exceeded. The state space is represented by the relative position of the four predators to the prey. All values of states are normalized to [1, 1] by dividing by the number of cells N. Tile coding [21] is used to force a generalization with 8 tilings and tile-width 0.5 to reduce the number of states. A capture results in a reward of 1 for every predator. In all other situations, the reward is 0. Despite this environment's representational and mechanical simplicity, it is still capable of presenting complex cooperative behaviors for MARL.

5.2.2 Parameters setup

PP game has one standard metric for performance evaluation: *Time to Goal* (TG) is the number of steps that four predators take to catch the prey. At every 100 training episodes, the TG values of this period are averaged to get a more smooth value. We train 20,000 episodes in total to get 200 TG values for this run, and all experiments

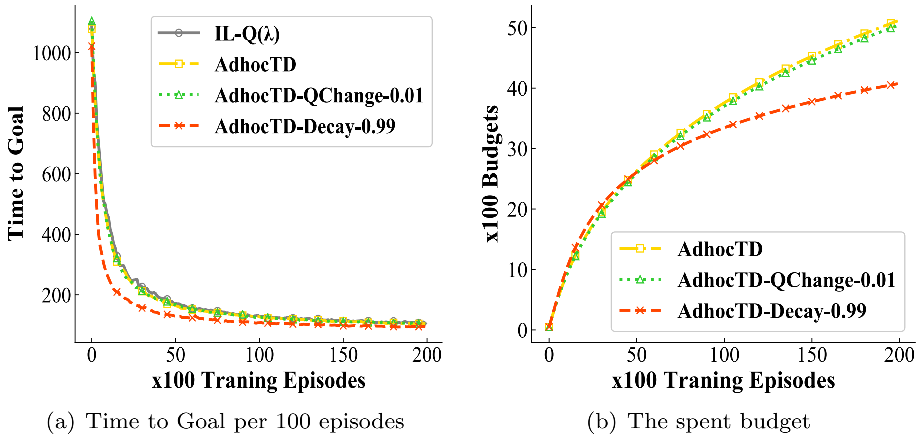


Fig. 3 TG and budget consumption of IL-Q(λ), AdhocTD and AdhocTD-QChange/Decay when $b_{ask} = b_{give} = 5000$, $\eta = 0.01$, and $\rho = 0.99$

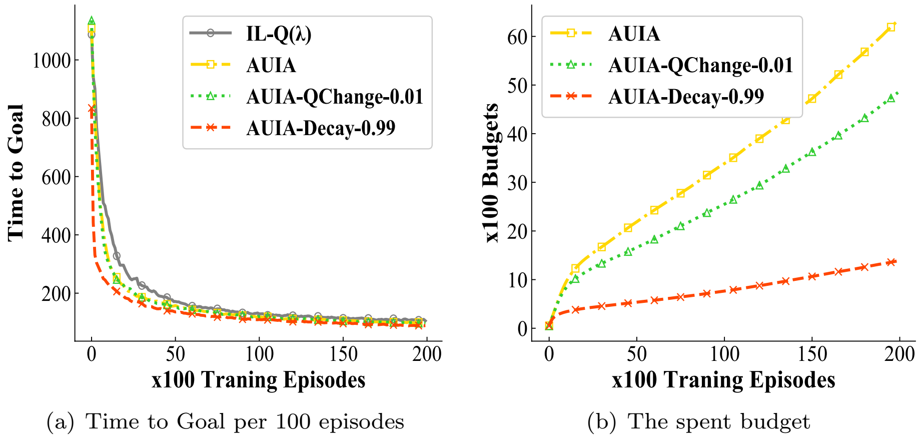


Fig. 4 TG and budget consumption of IL-Q(λ), AUIA and AUIA-QChange/Decay when $b_{ask} = b_{give} = 6000$, $\eta = 0.01$, and $\rho = 0.99$

are performed over 200 runs. For agents updating their Q-values, they use Q(λ) with a learning rate of $\alpha = 0.1$, a discount factor of $\gamma = 0.9$, and a decay rate of $\lambda = 0.9$. $\epsilon-$greedy with $\epsilon = 0.01$ serves as an exploration strategy for all agents. Similar to HFO, the parameters of AdhocTD and AUIA are selected to gain competitive performance. Thus, we choose $v_a = 0.2$ and $v_b = 1$ for AdhocTD and AdhocTD-QChange/Decay, and $t_a = 0.01$ and $t_g = 0.05$ for AUIA and AUIA-QChange/Decay, respectively. To compare the effect of different values of parameters, we set threshold $\eta = 0$, 0.01, and 0.03 for both AdhocTD-QChange and AUIA-QChange; and decay value $\rho = 0.9$, 0.99, and 0.999 for both AdhocTD-Decay and AUIA-Decay. For the budget setting, we use a sufficiently large budget to enable agents to ask for advice.

Table 2 Performance metrics for the agents in PP game (average of over 200 trials)

Agents	Initial	Last	Mean	Budget
IL-Q(λ)	1087.019	105.360	179.920	-
AdhocTD	1079.019	104.031	173.406	5113
AdhocTD-QChange ($\eta = 0$)	1109.262	102.907	176.157	5027
AdhocTD-QChange ($\eta = 0.01$)	1105.113	105.530	173.867	5036
AdhocTD-QChange ($\eta = 0.03$)	1129.243	104.310	175.901	4961
AdhocTD-Decay ($\rho = 0.9$)	1001.528	93.705	**127.653**	5982
AdhocTD-Decay ($\rho = 0.99$)	1021.499	94.667	136.923	4073
AdhocTD-Decay ($\rho = 0.999$)	1071.310	105.361	170.373	2163
AUIA	1108.693	96.665	157.976	6330
AUIA-QChange ($\eta = 0$)	1118.398	97.052	157.721	4791
AUIA-QChange ($\eta = 0.01$)	1135.733	97.509	156.734	4848
AUIA-QChange ($\eta = 0.03$)	1084.614	96.991	156.269	4857
AUIA-Decay ($\rho = 0.9$)	909.573	90.945	131.429	2296
AUIA-Decay ($\rho = 0.99$)	**835.043**	**88.890**	130.300	1386
AUIA-Decay ($\rho = 0.999$)	844.135	92.836	138.992	**862**

Lower values of Initial, Last, Mean and Budget are better. The best value of each column is shown in bold

5.2.3 Performance comparison

Figure 3 shows the *Time to Goal* (TG) and spent budget observed for IL-Q(λ), AdhocTD, AdhocTD-QChange with threshold $\eta = 0.01$ and AdhocTD-Decay with decay value $\rho = 0.99$. Notably, action reusing method AdhocTD-Decay has much lower TG values than all other algorithms, and consumes less budget. AdhocTD-QChange, which does not insist on previous advice for too long, has very similar results to AdhocTD. Figures 4a and 4b show, respectively, the TG metric and used advice budget achieved by AUIA and each action reusing algorithm. Unlike AdhocTD, AUIA achieves significantly lower TG than IL-Q(λ) before about episode 7000. AUIA-QChange with threshold $\eta = 0.01$ has similar performance to advising method AUIA while consuming much less budget. AUIA-Decay with decay value $\rho = 0.99$ finishes the experiment with less than 1500 budget used and achieves a significant speed-up of the learning.

Table 2 shows the first TG value (Initial), the last TG value (Last), the average TG values of the entire training episodes (Mean), and the spent budget (Budget) for independent learners, previous advising methods, and our proposed methods with different parameters. Notably, a proper value of threshold η, like 0.01 in AdhocTD-QChange or 0.03 in AUIA-QChange, can obtain lower TG values than other values of η. Since agents are learning and adjusting their policies together, increasing the threshold to a certain extent can help the agents reduce the impact of some non-optimal suggestions. As expected, when the decay value is 0.999, even with a smaller budget consumption, the performance is much worse than other parameters since the agents always follow the advice of others rather than learning by themselves.

5.3 Half field offense

5.3.1 Game description

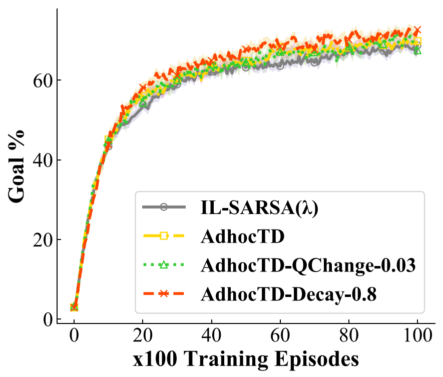
Half Field Offense (HFO) is a simulated robot soccer game [15], and we use the implementations released by Da Silva [9] with three players and one goalkeeper in the field. The goalkeeper adopts a fixed Helios Policy [1], which is obtained before the whole learning. The players are reinforcement learning agents and learn to shoot faster and more accurately. HFO is designed as an Episodic task. An episode starts when three payers and the ball are initialized with random positions on the field. The episode ends when either the players score a goal, the goalkeeper catches the ball, the ball leaves the field, or a time limit of 200 is exceeded. Note that each player can score a goal only if he possesses the ball, and otherwise, the option is to move. When a player possesses the ball, he can choose between four actions: *Shoot*, *Pass Near*, *Pass Far*, and *Dribble*. The players can benefit from cooperative behaviors: for example, one player passes the ball to another player for a better shot. A player's state is composed of the following observations:

1. Whether the player is in possession of the ball;
2. The proximity to the center of the goal;
3. The angle from one player to the goal center;
4. The largest angle of one player to the goal without blocking players;
5. The goal opening angle of the nearest (or farthest) partner.

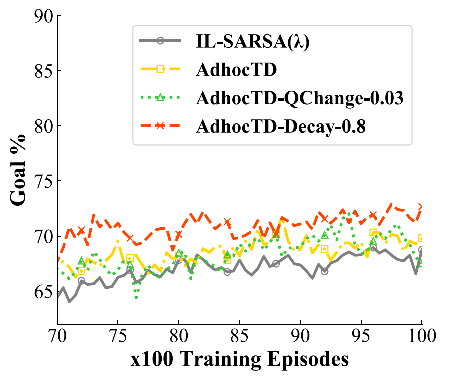
The state values are normalized in the range [1, 1] and discretized by Tile Coding [21] with 5 tilings and 0.5 tile size for simplifying the learning task. The players are awarded a reward of +1 when they score a goal, while they receive a reward of -1 if the ball is caught by the goalkeeper or out of bounds, encouraging them to find the goal as quickly as possible. In all remaining situations, the reward is 0.

5.3.2 Parameters setup

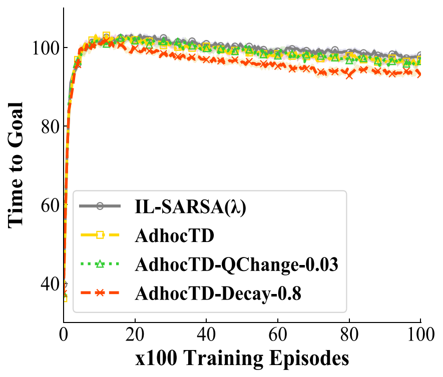
For evaluating the learning performance in this domain, we use two popular metrics: *Goal Percentage* (GP) is the percentage of testing episodes in which a goal is scored, and *Time to Goal* (TG) for HFO is the average number of steps that the players score a goal during testing episodes. All agents are trained for 10,000 episodes, with a pause at every 50 training episodes to perform 100 testing episodes. During testing, all players do not update their Q-values, ask for advice, or reuse previous advice. They utilize the currently best learned actions for every time step to obtain 200 GP (TG) values for that run. The procedure is repeated over 50 executions to draw the curve of averaged results. We performed all the experiments using SARSA(λ) with $\alpha = 0.1$, $\gamma = 0.9$, and $\lambda = 0.9$ as the algorithm to learn a policy in tasks. The ϵ-greedy is used as the exploration strategy of all players with $\epsilon = 0.1$. For the parameters v_a and v_b of AdhocTD, we use the same values adopted by Da Silva [9], which is also tested on HFO. Then we use $v_a = 0.5$ and $v_b = 1.5$ for AdhocTD, AdhocTD-QChange, and AdhocTD-Decay. As for AUIA, different values of thresholds have a great influence on the results. Finally, we choose $t_a = 0.01$ and $t_g = 0.03$, which consumes less budget while achieves similar results as other combinations of t_a and t_g. We test



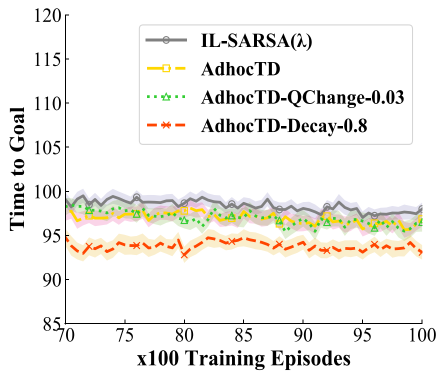
(a) Goal Percentage for 10,000 episodes



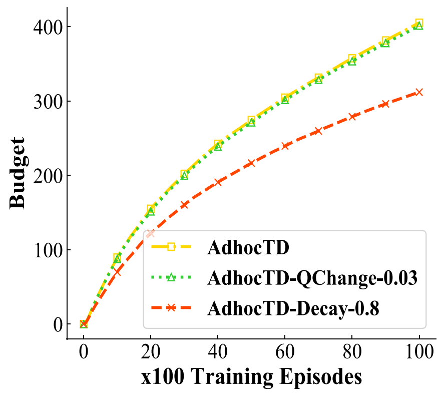
(b) Goal Percentage for last 3,000 episodes



(c) Time to Goal for 10,000 episodes



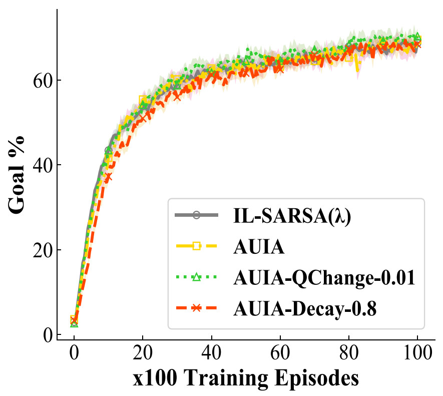
(d) Time to Goal for last 3,000 episodes



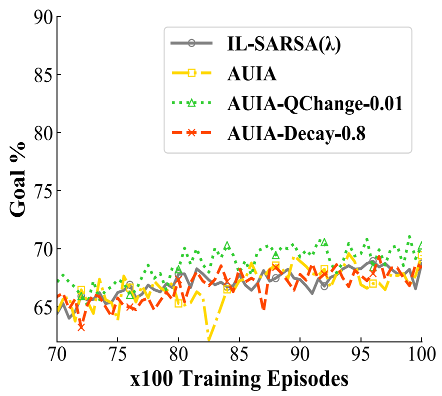
(e) The spent budget

Fig. 5 GP, TG and budget consumption of IL-SARSA(λ), AdhocTD and AdhocTD-QChange/Decay when $b_{ask} = b_{give} = 1000$, $\eta = 0.03$, and $\rho = 0.8$

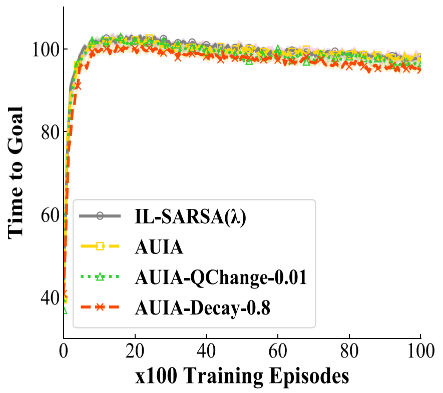
the proposed methods on different values of threshold η and decay value ρ. Thus we set: threshold $\eta = 0$, 0.01, and 0.03 for both AdhocTD-QChange and AUIA-QChange; decay



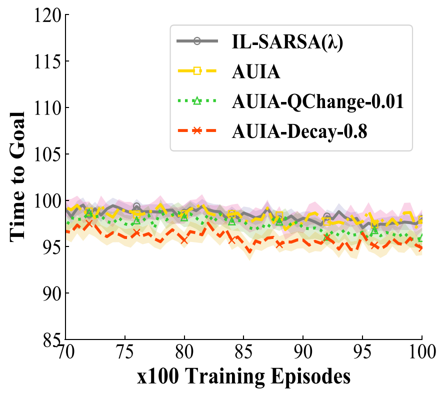
(a) Goal Percentage for 10,000 episodes



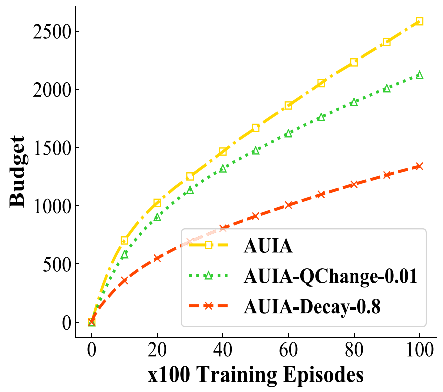
(b) Goal Percentage for last 3,000 episodes



(c) Time to Goal for 10,000 episodes



(d) Time to Goal for last 3,000 episodes



(e) The spent budget

Fig. 6 GP, TG and budget consumption of IL-SARSA(λ), AUIA and AUIA-QChange/Decay when $b_{ask} = b_{give} = 1000$, $\eta = 0.01$, and $\rho = 0.8$n

Table 3 GP metric for the agents in HFO (average of over 50 trials)

Agents	Initial	Last	Mean	Budget
IL-SARSA(λ)	2.990	68.740	58.421	-
AdhocTD	2.920	69.800	59.843	405
AdhocTD-QChange ($\eta = 0$)	3.210	68.490	59.847	404
AdhocTD-QChange ($\eta = 0.01$)	2.700	68.050	59.722	400
AdhocTD-QChange ($\eta = 0.03$)	3.300	67.510	59.796	402
AdhocTD-Decay ($\rho = 0.8$)	2.830	**72.670**	**61.701**	312
AdhocTD-Decay ($\rho = 0.9$)	3.320	70.840	61.256	283
AdhocTD-Decay ($\rho = 0.99$)	3.170	69.170	57.852	**176**
AUIA	**3.570**	69.450	58.167	2583
AUIA-QChange ($\eta = 0$)	3.100	70.080	59.235	2207
AUIA-QChange ($\eta = 0.01$)	2.680	70.300	59.430	2124
AUIA-QChange ($\eta = 0.03$)	3.220	69.020	59.311	2268
AUIA-Decay ($\rho = 0.7$)	3.200	66.230	56.332	1485
AUIA-Decay ($\rho = 0.8$)	3.310	68.440	56.671	1339
AUIA-Decay ($\rho = 0.9$)	3.080	66.800	54.769	1177

Higher values of Initial, Last and Mean are better. Lower values of Budget are better. The best value of each column is shown in bold

Table 4 TG metric for the agents in HFO (average of over 50 trials)

Agents	Initial	Last	Mean	Budget
IL-SARSA(λ)	39.700	97.990	99.104	-
AdhocTD	**36.220**	96.850	98.070	405
AdhocTD-QChange ($\eta = 0$)	39.850	97.130	98.086	404
AdhocTD-QChange ($\eta = 0.01$)	36.460	97.400	98.191	400
AdhocTD-QChange ($\eta = 0.03$)	37.490	96.510	98.242	402
AdhocTD-Decay ($\rho = 0.8$)	37.390	93.090	95.598	312
AdhocTD-Decay ($\rho = 0.9$)	39.960	**92.930**	**95.572**	283
AdhocTD-Decay ($\rho = 0.99$)	40.690	94.760	95.617	**176**
AUIA	39.480	97.890	98.775	2583
AUIA-QChange ($\eta = 0$)	38.900	96.770	97.992	2207
AUIA-QChange ($\eta = 0.01$)	36.890	95.950	98.113	2124
AUIA-QChange ($\eta = 0.03$)	38.260	95.790	97.938	2268
AUIA-Decay ($\rho = 0.7$)	38.260	96.640	97.551	1485
AUIA-Decay ($\rho = 0.8$)	40.800	94.860	96.583	1339
AUIA-Decay ($\rho = 0.9$)	39.890	94.840	96.506	1177

Lower values of Initial, Last, Mean and Budget are better. The best value of each column is shown in bold

value $\rho = 0.8$, 0.9, and 0.99 for AdhocTD-Decay, and $\rho = 0.7$, 0.8, and 0.9 for AUIA-Decay. For the budget setting, we follow AdhocTD to use sufficiently large budget to enable agents to ask for advice.

5.3.3 Performance comparison

Figure 5 shows the GP values, TG values, and consumed budget obtained by each algorithm. In Figures 5a and 5b, advising method AdhocTD achieves significant performance improvements on GP metric compared to IL-SARSA(λ). Based on AdhocTD, AdhocTD-Decay with decay value $\rho = 0.8$ obtains even higher GP values. Similar results can be observed in the TG metric. As shown in Figure 5c and 5d, after around 2000 episodes, AdhocTD-Decay starts to show better performance. The spent advice budget is recorded in Figure 5e. AdhocTD-Decay achieves significant improvements while spending the minimum budget.

Figure 6 shows the performance evaluated by GP, TG, and the used budget for IL-SARSA(λ), AUIA, AUIA-QChange with threshold $\eta = 0.01$, and AUIA-Decay with decay value $\rho = 0.8$. In Figure 6a and 6b with GP metric, we can see that AUIA performs worse than IL-SARSA(λ) before about 2000 episodes, which probably means that AUIA is not an ideal advising mechanism for HFO. Therefore, the performance of AUIA-Decay is not improved as expected. However, AUIA-QChange still achieves significant improvements roughly around after 4000 episodes. As for the TG shown in Figures 6c and 6d, both AUIA-QChange and AUIA-Decay have lower TG values than other methods. Figure 6e shows the consumed budget of each algorithm. Compared to AUIA, AUIA-Decay can save about half of the budget, while AUIA-QChange consumes higher budget than AUIA-Decay but lower than AUIA.

The first GP/TG value (Initial), the last GP/TG value (Last), the average GP/TG value of the whole training episodes (Mean), and the spent budget (Budget) for all experiments are shown in Tables 3, 4. We can see that when the decay value ρ increases, both AdhocTD-Decay and AUIA-Decay consume less budget. *Q-Change per Step* and *Decay Reusing Probability* can achieve better performance with selected critical parameters, i.e., threshold η and decay value ρ. Notably, the performance of our proposed methods depend on the fundamental advising mechanism. Nevertheless, AUIA-QChange surpasses the performance of AUIA and IL-SARSA (λ) on both GP and TG metrics.

6 Conclusion and future work

In this research, we address how to use and reuse advice more effectively in the teacher–student paradigm, which is particularly suitable under communication budget constraints. Our proposed framework, MBTSF, is extended from previous advising methods, in particular, for generating suggestions for reuse next time. With our proposed methods, when an agent encounters a state advised before, it either reuses previous advice, asks for new advice, or selects an action with a usual exploration strategy.

Table 5 summarizes the difference in the game characteristics used in our experiments. The effectiveness of MBTSF depends on the mechanism for generating advice. When the advising mechanism performs poorly, such as worse than the independent learner, MBTSF cannot guarantee that it performs better than the independent learner. But overall, compared with traditional teacher–student frameworks, MBTSF still performs better.

The effect of MBTSF also depends on the teacher's strategy. In multi-agent games like Predator–Prey and HFO, the teacher's strategy is adjusted and changed. In comparison, for the single-agent game Mario, the teacher is assumed by an agent that has been trained,

Table 5 Comparison of experiments

Comparison	Mario	Predator–Prey	Half field offense
Game type	Single-agent game	Multi-agent collaborative game, with each agent receiving the same reward	
Learner exchange	Independent learners: there is no communication among student agents, and agents only communicate the current state and suggested actions for teacher–student exchange		
Roles	Fixed teacher and student roles	The roles of students and teachers are not fixed, i.e., every agent has the opportunity to be a student or teacher	
Learning	Only the Mario agent learns in the environment, and it can only take the student's role. Another agent that has been trained in advance acts the teacher's role	Four hunters learn how to catch preys and suggest actions to one another	Three players learn to shoot and suggest actions to one another
Teacher's policy	The teacher's strategy has been fixed. The corresponding experimental effect is due to only the independent learner	The teacher's strategy is not fixed and is still being adjusted and learned	
AdhocTD, AdhocTD-QChange and AdhocTD-Decay	AdhocTD was originally used in multi-agent learning, where each agent can take the role of a teacher or a student. In this article, we fix the roles of teacher and student to enable AdhocTD to be used in Mario games, i.e., the trained agent act as the teacher to advise the Mario agent. Since AdhocTD decides when to ask and provide suggestions based on the number of times the agent visits the state (and the Q-value), it is necessary to record the number of times the teacher visits each state and the Q-value after training.	As Predator–Prey is a multi-agent game, AdhocTD can be used directly on Predator–Prey.	HFO was originally used to test the effect of AdhocTD. So, we directly use the settings of AdhocTD

Table 5 (continued)

Comparison	Mario	Predator–Prey	Half field offense
AUIA, AUIA-QChange and AUIA-Decay	AUIA was first used in an environment where the roles of teachers and students were fixed. In AUIA, the teacher decides when to provide advice based on the Q-value learned. So, we need to record the corresponding Q-value of each state after the teacher training	In multi-agent learning, the roles of teacher and student are not fixed, and each agent is learning. To adjust AUIA to apply multi-agent learning, the only change required is that the agent no longer records its own Q value, but when the student asks, it will decide whether to provide advice based on the Q value it learned	
Experiment result of AdhocTD	The effect of AdhocTD is worse than that of Independent Learner. In AdhocTD-QChange, when $\eta = 0$, its effect is better than AdhocTD and Independent Learner, and it costs less budget. When $\rho = 0.8$, AdhocTD-Decay achieved the best effect (highest ARE value) in all experiments	The effect of AdhocTD is better than than of Independent Learner. When $\eta = 0.01$, the effect of AdhocTD-QChange is similar to that of AdhocTD, but it costs less budget. In AdhocTD-Decay, all the values of ρ have obtained lower TG values. When ρ is 0.99, it is much better than AdhocTD and AdhocTD-QChange and consumes less budget	AdhocTD is superior to Independent Learner in both GP and TG. AdhocTD-QChange has the same effect as AdhocTD on GP and TG. When $\rho = 0.8$ and 0.9, AdhocTD-Decay is better than AdhocTD on TG and GP, and consumes less budget. When $\rho = 0.99$, the effect on GP is worse than Independent Learner and AdhocTD, but it consumes less budget
Expriment result of AUIA	The effect of AUIA is worse than that of Independent Learner and AdhocTD. In comparison, when $\eta = 0$, AdhocTD-QChange is better than AUIA and consumes much less budget. In AdhocTD-Decay, compared to AUIA and AdhocTD-QChange, all the values of ρ have achieved better results on ARE	The effect of AUIA is significantly better than that of Independent Learner and AdhocTD. AUIA-QChange has the same effect as AUIA under different η values but consumes less budget. The effect of AUIA-Decay is better than AUIA and AUIA-QChange, and when ρ is 0.999, the budget consumed is much lower than all other methods	The effect of AUIA is worse than Independent Learner on GP, and slightly better for Independent Learner on TG. AUIA-QChange is better than AUIA and Independent Learner on GP and TG, and consumes less budget. AUIA-Decay is worse than AUIA on GP but consumes less budget, and performs better than AUIA on TG

which makes the learning method based on reusing advice more effective. Notably, reusing suggestions not only increase the learning speed, but also consume less budget. Long-term adherence to the teacher's previous suggestions, such as using a larger ρ value in *Decay Reusing Probability*, may consume much less budget, but may also be harmful to learning because it cannot follow the teacher's learning in time.

Despite the variety of settings, our experiments show that reusing advice achieves significantly better performance than the advising methods without specifying advice reusing in all the scenarios. Moreover, when all agents are learning and adapting their policies, they need to explore alternatives while reusing the advised actions so that they may outperform their teachers. Last but not least, insisting on non-optimal teachers' advice for too long may even hinder the overall learning process.

For future work, we consider filtering the influence of bad advice, and learning a model to represent the shared knowledge from teachers. A student can also learn how to choose to reuse advice following the exploration strategy or ask for advice. Such a decision can be included in a high-level advising framework like LeCTR [19]. We would also consider using student feedback and teacher reputation [5] in our future framework. Besides, the influence of different budgets on advising and advice reusing process can be investigated.

Acknowledgements This work was supported by National Natural Science Foundation of China (62076100), and Fundamental Research Funds for the Central Universities, SCUT (D2210010, D2200150, and D2201300), the Science and Technology Planning Project of Guangdong Province (2020B0101100002).

References

1. Akiyama, H. (2012). Helios team base code.
2. Amir, O., Kamar, E., Kolobov, A., & Grosz, B. J. (2016). Interactive teaching strategies for agent training. In *Proceedings of the twenty-fifth international joint conference on artificial intelligence (IJCAI)* (pp. 804–811).
3. Barto, A. G., Thomas, P. S., & Sutton, R. S. (2017). Some recent applications of reinforcement learning. In *Proceedings of the 18th Yale workshop on adaptive and learning systems*.
4. Brys, T., Nowé, A., Kudenko, D., Taylor, M. (2014). Combining multiple correlated reward and shaping signals by measuring confidence. In *Proceedings of 28th AAAI conference on artificial intelligence* (pp. 1687–1693).
5. Chiu, D. K. W., Leung, H. F., & Lam, K. M. (2009). On the making of service recommendations: An action theory based on utility, reputation, and risk attitude. *Expert Systems with Applications, 36*(2), 3293–3301.
6. Claus, C., & Boutilier C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *The national conference on artificial intelligence* (pp. 746–752)
7. Clouse, J. A. (1996). *On integrating apprentice learning and reinforcement learning*. PhD thesis, University of Massachusetts
8. da Silva, F. L., & Costa, A. H. R. (2019). A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research, 64*, 645–703.
9. da Silva F. L., Glatt, R., & Costa, A. H. R. (2017). Simultaneously learning and advising in multiagent reinforcement learning. In *Proceedings of the 16th international conference on autonomous agents and multiagent systems* (pp. 1100–1108).
10. Felipe Leno da Silva, Pablo Hernandez-Leal, Bilal Kartal, and Taylor, M. E. (2020) Uncertainty-aware action advising for deep reinforcement learning agents. In The Thirty-Fourth AAAI Conference on Artificial Intelligence (pp.5792–5799
11. Felipe Leno da Silva, Matthew E. Taylor, and Anna Helena Reali Costa (2018) Autonomously reusing knowledge in multiagent reinforcement learning. In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (pp.5487–5493
12. Fachantidis, A., Taylor, M. E., & Vlahavas, I. P. (2017). Learning to teach reinforcement learning agents. *Machine Learning and Knowledge Extraction, 1*, 21–42.

13. Ilhan, E., Gow, J., & Liebana, D. P. (2019) Teaching on a budget in multi-agent deep reinforcement learning. arXiv:1905.01357
14. Karakovskiy, S., & Togelius, J. (2012). The Mario AI benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games, 4*(1), 55–67.
15. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., & Matsubara, H. (1997). Robocup: A challenge problem for AI. *AI Magazine, 18*, 73–85.
16. Kober, J., Bagnell, J.A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research, 32*, 1238–1274.
17. Matignon, L., Laurent, G. J., & Le Fort-Piat, N. (2012). Independent reinforcement learners in cooperative markov games: A survey regarding coordination problems. *Knowledge Engineering Review, 27*, 1–31.
18. Oliehoek, F. A., & Amato, C. (2016). *A concise introduction to decentralized POMDPs* (1st ed.). Springer: New York.
19. Omidshafiei, S., Kim, D.-K., Liu, M., Tesauro, G., Riemer, M., Amato, C., Campbell, M., How, J. P. (2019). Learning to teach in cooperative multiagent reinforcement learning. In *The thirty-third AAAI conference on artificial intelligence* (pp. 6128–6136).
20. Rummery, G. A., & Niranjan, M. (1994). On-line q-learning using connectionist systems. Technical report cued/f-infeng/tr 166, Cambridge University Engineering Department.
21. Sherstov, A. A., & Stone, P. (2005). Function approximation via tile coding: Automating parameter choice. In *Proceedings symposium on abstraction, reformulation, and approximation (SARA-05), Edinburgh, Scotland, UK*
22. Suay H. B., Brys T., Taylor, M. E., & Chernova S. (2016). Learning from demonstration for shaping through inverse reinforcement learning. In *Proceedings of the 2016 international conference on autonomous agents and multiagent systems)* (pp. 429–437).
23. Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction* (1st ed.). Cambridge, MA: MIT Press.
24. Taylor, M. E., Carboni, N., Fachantidis, A., Vlahavas, I. P., & Torrey, L. (2014). Reinforcement learning agents providing advice in complex video games. *Connection Science, 26*(1), 45–63.
25. Torrey, L., & Taylor, M. E. (2013). Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of 12th the international conference on autonomous agents and multiagent systems* (pp. 1053–1060).
26. Wang, Y., Lu, W., Hao, J., Wei, J., & Leung, H. f. (2018). Efficient convention emergence through decoupled reinforcement social learning with teacher–student mechanism. In *Proceedings of the 17th international conference on autonomous agents and multiagent systems* (pp. 795–803).
27. Wang, Z., & Taylor. M. E. (2017). Improving reinforcement learning with confidence-based demonstrations. In *Proceedings of the twenty-sixth international joint conference on artificial intelligence (IJCAI)* (pp. 3027–3033).
28. Watkins, C. J. C. H., & Dayan, P. (1992). Technical note: Q-learning. *Machine Learning, 8*, 279–292.
29. Zhan, Y., Bou-Ammar, H., & Taylor, M. E. (2016). Theoretically-grounded policy advice from multiple teachers in reinforcement learning settings with applications to negative transfer. In *Proceedings of the twenty-fifth international joint conference on artificial intelligence* (pp. 2315–2321).
30. Zhu, C., Cai, Y., Leung, H.-f., & Hu, S. (2020). Learning by reusing previous advice in teacher–student paradigm. In: A. El Fallah Seghrouchni, G. Sukthankar, B. An, and N. Yorke-Smith (Eds.), *Proceedings of the 19th international conference on autonomous agents and multiagent systems, AAMAS '20, Auckland, New Zealand, May 9–13, 2020. International foundation for autonomous agents and multiagent systems, 2020* (pp. 1674–1682).
31. Zimmer, M., Viappiani, P. & Weng, P. (2014). Teacher–student framework: A reinforcement learning approach. In *AAMAS workshop*.