Journal of Machine Learning Research 24 (2023) 1-34 Submitted 4/23; Revised 11/23; Published 11/23

Boosting Multi-agent Reinforcement Learning via Contextual Prompting

Yue Deng DEVINDENG@ZJU.EDU.CN
Zirui Wang ZISEOIWONG@ZJU.EDU.CN
Xi Chen CHAN_XI@ZJU.EDU.CN
Yin Zhang * ZHANGYIN98@ZJU.EDU.CN
College of Computer Science and Technology
Zhejiang University
Hangzhou, China

Editor: Scott Niekum

Abstract

Multi-agent reinforcement learning (MARL) has gained increasing attention due to its ability to enable multiple agents to learn policies simultaneously. However, the bootstrapping error arises from the difference between the estimated Q value and the real discounted return and accumulates backward through dynamic programming iterations. This error can become even larger as the number of agents increases, due to the exponential growth of agent interactions, resulting in infeasible learning time and incorrect actions during early training steps. To address this challenge, we observe that previously collected trajectories are useful contexts, model them using a contextual predictor to yield the next action and observation, and use the contextual predictor to replace the Q value function or utility function during the early training phase. Furthermore, we employ a joint-action sampling mechanism to restrict the action space and dynamically select policies from the vanilla utility network and those from the contextual trajectory predictor to perform rollout processes. By reasonably constraining the action space and rollout process, we can significantly accelerate the algorithm training process. Our framework applies to various value-based MARL methods in both centralized training decentralized execution (CTDE) and non-CTDE scenarios where agents are accessible (non-accessible) to global states during the training process. Experimental results on three tasks, Spread, Tag, and Reference, from the Particle World Environment (PWE) show that our framework significantly accelerates the training process of existing state-of-the-art CTDE and non-CTDE MARL methods, while also competing with or outperforming their original versions.

Keywords: MARL, Training Boosting, Joint-action Sampling, Contextual Predictor, Joint-policy Collaboration

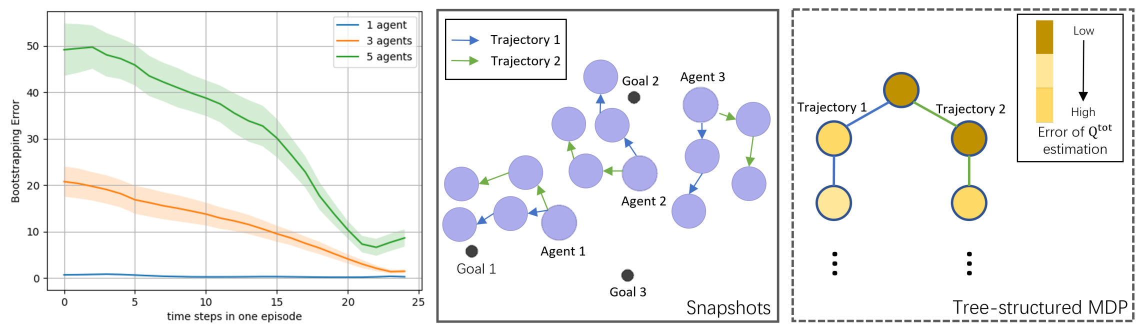
1. Introduction

Recent advances in Reinforcement Learning (RL) have led to significant progress in solving complex control systems, such as robotics and Atari games (Mnih et al., 2013; Silver et al., 2017). Multi-agent reinforcement learning (MARL) extends RL to a multi-agent setting,

∗. Corresponding Author: Yin Zhang.

which has broad real-world applications, including autonomous vehicle teams (Cao et al., 2012) and sensor networks (Zhang and Lesser, 2011). Various MARL methods have been proposed to address value decomposition (Sunehag et al., 2017; Rashid et al., 2018, 2020; Wang et al., 2020a) or cooperative exploration (Yang et al., 2020; Mahajan et al., 2019; Wang et al., 2020b). Among them, value-based MARL methods (Sunehag et al., 2017; Son et al., 2019; Wang et al., 2019) have achieved state-of-the-art performance on challenging tasks such as StarCraft II (Samvelyan et al., 2019).

However, bootstrapping-based Q-learning methods used in RL can suffer from a pathological interaction between the function approximation and the data distribution. When training the Q-function in standard Q-learning methods, online data collection should induce corrective feedback, and new data should correct mistakes in old predictions. Such feedback may be absent in dynamic programming methods like Q-learning, leading to instability, sub-optimal convergence, and poor results when learning from noisy, sparse, or delayed rewards in early time steps (Kumar et al., 2020). This phenomenon is further compounded in multi-agent scenarios, as shown in Figure 1a, where the accumulated bootstrapping error becomes much more critical as the number of agents grows. The large joint action space results in much larger exploration spaces, which are needed to induce corrective feedback. Additionally, the training steps required for convergence of the original MARL methods increase significantly as a result of the exponential growth of agent interactions, limiting the prospects for the application of MARL algorithms.



(a) (b) (c)

Figure 1: (a) Bootstrapping errors on each time step with different numbers of agents. X-axis is the time step in an episode, and the y-axis represents the bootstrapping error in each time step. The bootstrapping errors are defined as the difference between the estimated value of a certain state and the expected ground truth value of that state. In this figure, the bootstrapping values are calculated by deducing the estimated Q value of each state from policy networks and the average discounted return of that state within interaction trajectories among 5 seeds. (b) Three blue agents should move towards three black goals, and the graph is the snapshot of two different trajectories of each agent's moving trace. (c) The graph is a tree-structured POMDP representing the two corresponding trajectories with the bootstrapping error of each node. The bootstrapping errors are small when agents are nearer the target goals, and in contrast large when agents are far away. These errors are accumulated in the earlier states, and errors are back-propagated through branches in tree-structured POMDP problems.

To address these challenges, we propose a novel framework with context prompting, an MARL framework consisting of joint-action sampling, and context trajectory prediction mechanisms. Instead of generating actions from a Q-function, we generate actions from a self-supervised contextual predictor trained on historical agent interaction data from the replay buffer. Inspired by sampling-based Muzero (Hubert et al., 2021) and trajectory prediction (Tang et al., 2021), we develop a contextual trajectory tree based on historical trajectories from the replay buffer in a Monte-Carlo Tree structure to provide a sequence of observation transitions as contextual prompts for each agent. We also propose two variants of contextual prompt generators based on tree and pool implementations. To make our contextual prompting framework more general, we use a supervised Multi-Layer Perceptron (MLP) with Gated Recurrent Units (GRU) cells, which we call the Contextual Predictor, to learn the data in the trajectory model. The contextual predictor and the utility network in the MARL methods form a collaborative controller that interacts with the environment during the rollout processes. We evaluated our approach in the particle world environment (PWE) and demonstrated that our framework is sufficiently compatible with existing value-based state-of-the-art CTDE (Kraemer and Banerjee, 2016) MARL methods, e.g. QMIX (Rashid et al., 2018) and non-CTDE methods, e.g. IQL (Tampuu et al., 2017). More importantly, MARL methods adapted to our framework are able to compete with or outperform their original methods in terms of the final performance with limited computational resources and a relatively small number of training steps.

In summary, the main contributions of this work are as follows:

1. We propose a contextual prompting framework that can adapt CTDE and non-CTDE value-based MARL methods. Our prompting method is also compatible with value-based single-agent scenario algorithms.

2. Our contextual prompting method can compete with or outperform original value-based MARL methods in terms of final performance while significantly accelerating convergence speed according to the experimental results.

3. We describe the feasibility of our framework from theoretical perspectives and validate our framework empirically by sufficient experiments on multiple scenarios in PWE environments.

The organization of this paper is as follows: Section 2 introduces recent work related to our method, including MARL, contextual prompting, and trajectory forecasting. Section 3 introduces the basic concepts and notations related to our approach. Section 4 describes the architecture, contextual prompting methods, and training pipeline of our approach. Section 5 describes our setup of the experiment, the experimental results, and the discussion based on the results. Conclusions and future works are mentioned in Section 6.

2. Related work

Existing studies on multi-agent reinforcement learning (MARL) have focused mainly on using a central value function to guide the learning process of individual value functions and studying MARL methods in a centralized training decentralized execution (CTDE) setting

(Lowe et al., 2017; Sunehag et al., 2017; Rashid et al., 2018; Wang et al., 2019). However, there are relatively few works investigating the acceleration of the training process of existing MARL methods through constrained trajectory modeling or joint-action sampling. Previous studies discuss the impact of whether to relax monotonic constraints (Rashid et al., 2020; Mahajan et al., 2019) or whether to share gradients or parameters on the training efficiency of MARL methods (Chu and Ye, 2017; Christianos et al., 2020, 2021; Terry et al., 2020; Kuba et al., 2021).

In standard reinforcement learning, how to accelerate the training process by improving sampling efficiency and state representation learning has been well studied (Nachum et al., 2018; Buckman et al., 2018; Du et al., 2019; Laskin et al., 2020; Kostrikov et al., 2020; Ye et al., 2021b; Yarats et al., 2021). However, in multi-agent settings, the exponential growth of the agent interactions and joint-action space with the number of agents makes it infeasible to enumerate every state or joint-action, which brings challenges for MARL training acceleration. Wang et al. (2023) examines the use of distribution matching to facilitate the coordination of independent agents. Oh et al. (2018) proposed that exploiting good past experiences can indirectly drive deep exploration and is competitive to state-of-the-art count-based exploration methods in single-agent scenarios.

In some real-world situations, metadata and additional information about a task may inform relations between multiple tasks. An efficient approach to knowledge transfer is through the use of multiple context-dependent, composable representations shared across a family of tasks (Sodhani et al., 2021). Fu et al. (2021) proposes an approach toward an effective context for meta-reinforcement learning by contrastive learning. Xu et al. (2021) proposed a context-based meta-reinforcement learning algorithm by generating an efficient task encoder. Seyed Ghasemipour et al. (2019) applied context-conditional policies to meta inverse reinforcement learning. Unlike the above works for single-agent scenarios, we apply contextual prompting methods on multi-agent tasks by simple implementations.

Another line of research is multi-agent trajectory forecasting. This task takes the observed trajectories of multiple agents as input and outputs a predicted trajectory for each agent (Liang et al., 2019; Kosaraju et al., 2019), and has applications in autonomous vehicles(Ye et al., 2021a; Gilles et al., 2021), drones (Xiao et al., 2019), and industrial robots (Rösmann et al., 2017). Unlike the above works, in this work trajectory modeling is used to accelerate the training process of existing MARL methods, rather than measuring dynamic uncertainty or modeling the interaction mechanisms between agents.

Monte-Carlo tree search (MCTS) (Coulom, 2006) is a heuristic search algorithm that combines classic tree search implementations alongside the machine learning principles of reinforcement learning. MCTS works by repeatedly sampling the game tree and simulating games from different starting positions. The algorithm then uses these simulations to estimate the value of different moves and to select the best move to make. Instead of expanding a decision tree from one state, the Monte-Carlo Trajectory Tree (MCT^2) in this paper stores the historical interaction trajectories and models them similarly to MCTS. Another difference is that MCTS makes a decision in one step, but MCT^2 generates a whole trajectory.

To the best of our knowledge, CTDE can be seen as an effective approach to accomplishing this objective. Furthermore, our framework provides a significant acceleration of the training process of current value-based state-of-the-art CTDE and non-CTDE MARL methods, promising to facilitate efficient training in large-scale multi-agent environments, an important step towards real-world multi-agent applications.

3. Background

In this paper, we formulate the multi-agent system as a decentralized partially observable Markov decision process (Dec-POMDP) (Ong et al., 2009), which can be defined as $G = \langle S, A, P, r, \Omega, N, \gamma \rangle$. At each time step t, $s \in S$ describes the true state of the environment, every agent $i \in \mathcal{N} := \{1, ..., N\}$ draws individual independent observations $o^i \in \Omega$ according to the observation function $O(s, a) : S \times A \rightarrow \Omega$. Meanwhile, each agent chooses an action $a^i \in A^i$ [1] on which it conditions a stochastic policy $\pi_{\theta^i}(o^i, a^i) \in [0, 1]$. The joint policy $\boldsymbol{\pi}_\theta = [\pi_{\theta^0}, ...\pi_{\theta^N}]$ is the vector of the individual agent policies.

In addition, $P(s' \mid s, \boldsymbol{a}) : S \times (A)^N \times S \rightarrow [0, 1]$ is the transition dynamics function. Each agent i obtains a shared reward r according to a reward function $R(s, a_{joint}) \in \mathbb{R}$, where $a_{joint} := (a^1, a^2, \cdots, a^N)$ is the concatenation of actions of all agents[2]. The reward function is conditioned on the true state and the joint actions of the agents.

The sequence of observations from agent i, $o^i_{0:t} = (o^i_0, o^i_1, ..., o^i_t)$, forms the history of that agent, in which t is the time step during the process. In a complete period, agent i obtains its own total expected accumulated rewards $\mathcal{R}^i = \sum_{t=0}^{T} (\gamma^t) r^i_t$ in which γ is the discount factor and T is the time horizon and r^i_t is the reward agent i receives at time step t.

Value-based methods need to learn the optimal state-action value function $Q_\star(s_t, a_t)$. Deep Q networks replace $Q_\star(s_t, a_t)$ by a parametric network $Q(s_t, a_t|\omega)$ and optimize the network according to the loss function of the squared TD error:

$$L(w) = \left[Q\left(s_t, a_t|w\right) - \widehat{y_t}\right]^2, \tag{1}$$

where $\widehat{y_t} = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \omega^-)$. ω^- are the parameters of a target network that are periodically copied from ω and kept constant for a number of iterations.

In IQL algorithm (Tampuu et al., 2017), each agent learns action policy π^i individually according to their observation o^i as deep Q-Learning (Sutton and Barto, 2018) algorithm for single-agent RL tasks. In contrast, VDN algorithm (Sunehag et al., 2018) decomposes Q_{tot} into Q^i for each agent and the total value function can be obtained by $Q_{tot} = \sum_{i=1}^{N} Q^i$. Based on VDN algorithm, QMIX (Rashid et al., 2018) introduces a monotonic restriction $\frac{\partial Q_{tot}}{\partial Q^i} \geq 0, \forall i \in \{1, 2, \cdots, n\}$ implemented by a hyper-network. The mixing network is a feed-forward neural network that takes the agent network outputs as input and mixes them monotonically, producing the values of Q_{tot}. To enforce the monotonicity constraint, the weights of the mixing network are restricted to be non-negative, which allows the mixing network to approximate any monotonic function arbitrarily closely. The loss function for QMIX within a batch of historical transitions with its size b becomes:

$$\mathcal{L}(\omega) = \sum_{i=1}^{b} [(Q_{tot}(\boldsymbol{\tau}, \mathbf{a}, \mathbf{s}|\omega) - y^i_{tot})^2]. \tag{2}$$

Meanwhile, the QMIX algorithm is also supposed to follow the IGM (Son et al., 2019) regulation, given below:

1. A^i means the collections of available actions of agent i
2. $a_{joint} \in \mathcal{A}$, where $\mathcal{A} := A^1 \times A^2 \times \cdots \times A^N$

$$\arg\max_u Q_{tot}(o, \mathbf{a}) = \begin{pmatrix} \arg\max_{a_1} Q^1(o^1, a^1) \\ \vdots \\ \arg\max_{a_n} Q^n(o^n, a^n) \end{pmatrix}.$$

4. Method

In this section, we introduce the overall architecture of our framework and describe the contextual prompting methods including the obtaining of the contextual prompt, multi-agent action sampling, and the training of contextual predictor. In the contextual prompt obtaining process, we establish a trajectory tree from data in the replay buffer in the Monte-Carlo Tree structure which in our paper we call it Monte-Carlo Trajectory Tree (MCT^2). Instead of selecting one action according to the largest value in MCT planning, our work models data as an MCT structure to provide contextual prompts for contextual predictor training. Additionally, the training pipeline and the pseudo-code of our framework are also provided in this section.

4.1 Architecture

In our framework, joint policies are collaboratively generated by the utility network of baseline MARL algorithms and our contextual predictor network. Specifically, actions generated from baseline MARL algorithms, such as QMIX, are conditioned on agents' observations and selected based on the maximum of $Q(o_t^i, a_t^i)$ values. The loss function is based on the TD error of $Q_{tot}(\mathbf{o}, \mathbf{a})$, and policies are generated from the utility network. Alternatively, our contextual predictor generates actions for the current time step and predicts the observations in the subsequent time step. The loss function of our contextual predictor is composed of the Cross-Entropy (CE) loss of discrete actions and the Mean Square Error (MSE) loss of predicted continuous observations. A switch function is used to control which actions, utility outputs, or contextual predictor outputs should be selected for the rollout process and described in subsection 4.3. The overall architecture of our framework is depicted in Figure 2.

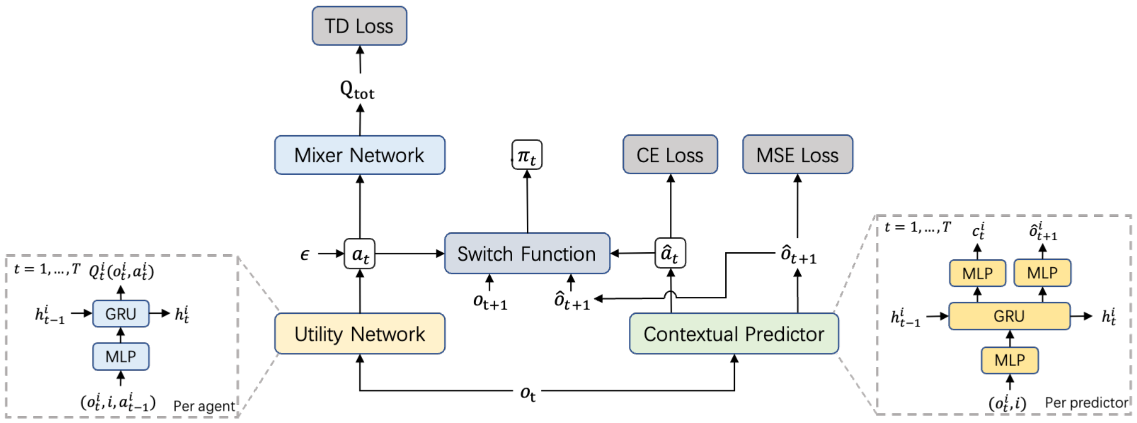


Figure 2: The architecture of our framework.

Figure 2 illustrates the overall architecture of our framework. The utility network, which is adapted from baseline MARL, is responsible for calculating $Q(o_t^i, a_t^i)$ for each agent's observation o_t^i and action a_t^i at time step t. The maximum Q-value is selected to determine the agent's action, which is controlled by a switch function. These Q-values are then fed into the mixer network, a hyper network that enforces the monotonic restriction in the QMIX algorithm. The detailed settings on the mixer network are from the QMIX paper Rashid et al. (2018). The utility network is trained after each rollout process. The interaction data with environment in each episode $[(s_0, \mathbf{a_0}, \mathbf{o_0}, r_0)...(s_T, \mathbf{a_T}, \mathbf{o_T}, r_T)]$ are stored sequentially inside a replay buffer. Transitions within trajectories are sampled from this replay buffer to train the utility networks and the mixer network.

Our contextual predictor takes the observation o_t^i of each agent as input and predicts the best action based on the trajectories that it has seen, $\hat{a}_t^i$, for the current time step, as well as predicting the observation of the next time step $\hat{o}_{t+1}^i$. These transition data can be sampled directly from the trajectory pool, the replay buffer which stores a number of interaction trajectories, or can be obtained from a tree-based data structure built from these transitions. In this paper, we first sample some trajectories from the replay buffer and model them as a Monte-Carlo Trajectory Tree to provide the current best routes for training the contextual predictor. The MCT^2 method is applied centrally and the sampled trajectories are used for training the contextual predictor individually per agent. The construction and the update of the trajectory tree are described in subsection 4.2.1. The selected routes are fed into the predictor network for training, which is described in subsection 4.2.2. The switch function controls the rollout action, whether it is the predicted action or the action selected by the utility network, depending on the precision between the predicted observation and the actual observation from the environment, $dist(\hat{o}_{t+1}^i, o_{t+1}^i)$. During the precision calculation, if the action is selected from the contextual predictor, the observation of the next time step o_{t+1} is generated from o_t and the predicted action $\hat{a}_t$. Once the utility network takes over the decision-making tasks, the o_{t+1} is generated from o_t and the action from utility network a_t in the subsequent time steps (subsection 4.3).

It is worth noting that the observation o_t^i is used as input for both the utility network and the predictor network. This ensures that the context used by our contextual predictor is also fed into the utility network, which aligns the data used for training.

4.2 Contextual Prompt

In this subsection, we introduce the multi-agent action sampling principle and the obtaining method of contextual prompts. We construct a Monte-Carlo Trajectory Tree (MCT^2) from data sampled from the replay buffer to provide the current best routes for training our contextual predictor. We also describe the training details including the design of loss functions and their functionality.

4.2.1 Obtain contextual prompt

Action Sampling: During the construction of the MCT^2, the actions are sampled and the width of branches is constrained. Therefore, policies that are evaluated and improved over sampled action subsets should also converge to optimal results. Hubert et al. (2021) proposed a sample-based policy evaluation and improvement method for single-agent systems, especially those with complex action spaces. By projecting $\mathcal{I}_\pi$ back onto the realizable policies space, we can compute the improved policy by sampling from the actions spaces.

For a policy $\pi : S \to \mathcal{P}(\mathcal{A})$, where the state space S and the action space $\mathcal{A}$, its improved policy $\mathcal{I}\pi : S \to \mathcal{P}(\mathcal{A})$ satisfies $\forall s \in \mathcal{S}, v^{\mathcal{I}\pi}(s) \geq v^{\pi}(s)$. If $\mathcal{I}_\pi$ is completely accessible, it could be directly used for policy improvement by projected onto the space of realizable policies. However, the large action space $\mathcal{A}$ makes it feasible to compute an improved policy over a small subset of actions.

Ghosh et al. (2020) showed that the policy gradient algorithm can be thought of as having the following policy improvement operator: $\mathcal{I}_\pi \propto \pi(s,a)Q(s,a)$ where $Q(s,a)$ is the action-value function. A policy improvement operator is defined as action-independent if it can be written as $\mathcal{I}_\pi(a|s) = f(s,a,Z(s))$ where $Z(s)$ is a normalizing state dependent factor defined by $\forall a \in \mathcal{A}, f(s,a,z(s)) \geq 0$ and $\sum_a f(s,a,Z(s)) = 1$.

We combine the concepts of improved policy and policy operators and propose a sample-based action-independent Policy Improvement Operator:

$$\hat{\mathcal{I}}_\beta \pi(a_{joint}|s) = (\hat{\beta}/\beta)(a_{joint}|s) f(s, a_{joint}, \hat{Z}_\beta(s)) \tag{3}$$

where $(\hat{\beta}/\beta)(a_{joint}|s) \iff \hat{\beta}(a_{joint}|s)/\beta(a_{joint}|s)$. In detail, the joint-action set $\{a^i_{joint}\}$ are K actions sampled from the proposal distribution β and can be reconstructed into the empirical distribution: $\hat{\beta}(a|s) = \frac{1}{K}\sum_i \delta_{a_{joint}, a^i_{joint}}$, where δ_{a,a_i} represents the Kronecker delta function.

Hubert also proved that the distribution of sample-based Policy Improvement Operator can converge to the true policy improvement operator: $\lim_{K\to\infty} \hat{\mathcal{I}}_\beta \pi = \mathcal{I}\pi$.

Monte-Carlo Trajectory Tree Generation: Based on the convergence guarantees described above, we generate MCT^2 by the data sampled from the replay buffer. States $\mathbf{s}$ from the sampled data are classified into N_{clu} clusters for each time step. In our trajectory tree, we model actions $\mathbf{a}_t$ as branches and IDs of clusters as nodes in which we store states s_t and corresponding observations o_t. t not only represents the time step during the rollout process but also the depth of the node. A transition $(s_t, o_t, a_t, s_{t+1}, o_{t+1})$ can be regarded as a visit from a node that contains s_t at layer t to its child node that contains s_{t+1} at layer $t+1$ by action a_t. Additionally, a node also stores the expected return following a sample-based improved policy $\hat{\mathcal{I}}\pi$ in an environment starting from that node.

To construct the tree and train the cluster classifier, we randomly select $t_{interval}$ trajectories from the replay buffer and apply a clustering method following the idea of Expectation-Maximization(EM), iteration algorithms, such as KMEANS and DBSCAN:

$$\theta^{(it+1)} = \arg\max_\theta \sum_{z=1}^{N_{clu}} \log P(s,z|\theta) \times P(z|s,\theta^{(it)}) \tag{4}$$

Inside the formula above, θ is the parameter of the cluster model, N_{clu} is the maximum number of clusters, and z is the cluster to which an s belongs. Clustering algorithms may take several iterations for cluster centers to converge and it is the iteration times. After each Expectation phase iteration, the cluster is calculated by maximum probability: $p(z) = P(z|s,\theta)$. We suppose that the values of subsequent states generated from a certain

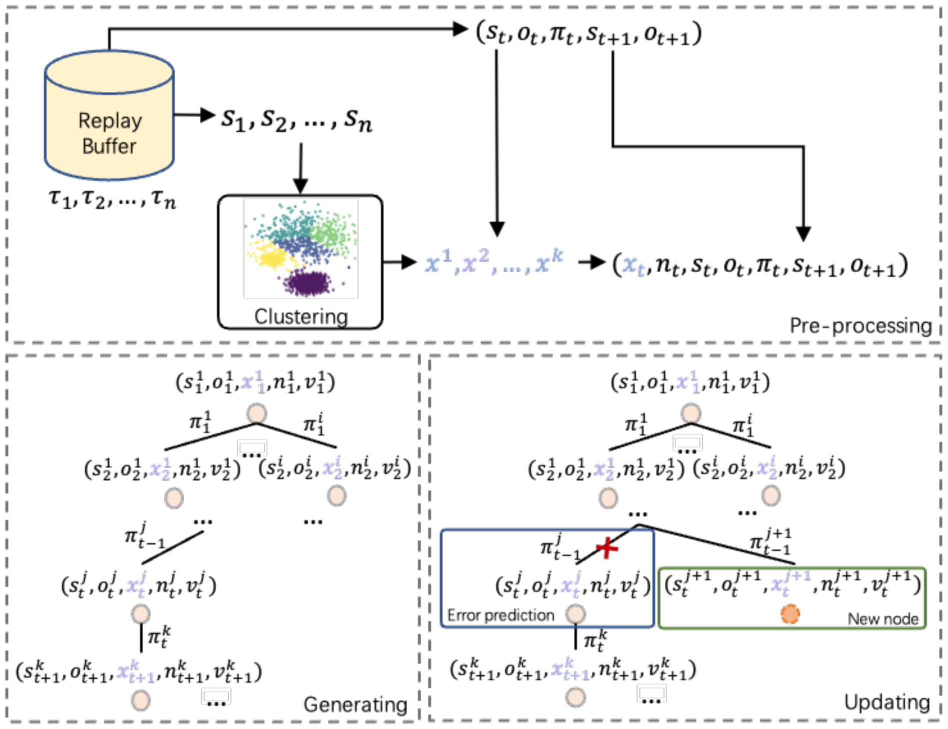


Figure 3: **Top:** Sample n trajectories τ from replay buffer and classify states s_i into k clusters ($x^1 \ldots x^k$). Each transition data belongs to one cluster. **Left:** MCT^2 generation based on sampled trajectories from replay buffer. Transitions in a trajectory can be reflected as a path from the root node to a leaf node. The superscript in this graph is the index of clusters in this layer and the subscript is the depth of the tree. **Right:** New node generation process when context predictor predicts wrong observation.

parent state and the chosen actions on those states are independent and they follow normal distributions. The expectation phase is the centroid state calculation from a cluster:

$$cs_i^{(it+1)} = \frac{1}{|C_i^{(it)}|} \sum_{s_j \in C_i^{(it)}} s_j. \tag{5}$$

Inside the formula above, $cs_i^{(it+1)}$ is the centroid state of cluster C_i in the $(it+1)$th iteration. s_j is the state that belongs to cluster C_i. For fast calculation, we store the size of a cluster instead of the states belonging to that cluster. Therefore, the calculation can be formulated as:

$$cs_i^{(it+1)} = \frac{1}{n+1}(cs_i^{(it)} \times n), \quad n = |C_i^{(it)}|. \tag{6}$$

Additionally, the maximization phase is the assignment of new states to their clusters by minimum distance calculation:

$$s \in \arg\min_{cs_i \in C_i} dist(s, cs_i), \quad i = 1, 2, \ldots n. \tag{7}$$

To address the possible diversity of starting clusters which results in multiple roots, we introduce a hidden initial node that serves as the parent node for all possible starting clusters. In this way, the trajectory forest is changed into a trajectory tree by a hidden hyper-node. When a new state is added to the tree, it is classified into a specific node, and the centroid of that node is updated accordingly.

To avoid overfitting and improve the efficiency of the tree, we propose a pruning operation to remove trajectory branches with high values but low visitation probabilities after each iteration. The pruned branch spaces are reserved for expansion in the following iterations, with the maximum number of branches and the number of pruned branches determined by hyperparameters. To address the issue of the policy shift, we adopt periodic reconstruction of our tree and train a new classifier by sampling a batch of data. This helps ensure that our tree and policy remain effective and adaptable to new scenarios.

Monte-Carlo Trajectory Tree Updating: When a new episode terminates, the cumulative reward is set as the $Q(c, a)$ value of the new node. Here, c represents the cluster node, and a represents action selection. During backpropagation, the values of the nodes on that path are updated according to the following formula:

$$Q_p(c, a) = \lambda Q_p(c, a) + (1 - \lambda)\mathbb{E}_{c\sim\mathcal{C},a\sim\mathcal{A}}[r_{p\to c} + \gamma Q_c(c, a)]. \tag{8}$$

Here, $Q_p(c, a)$ represents the value of the parent node during backpropagation, and $Q_c(c, a)$ represents the values of the children nodes from that parent node. $r_{p\to c}$ is the expectation of the historical rewards from the current parent node to its child node with action a. The value of λ is dynamically changed after expansion and is empirically defined as $1/n$.

After the rollout process in each episode, our framework selects or samples K trajectories from the MCT^2 and trains the contextual predictor. The tree is periodically destroyed and reconstructed every $t_{interval}$ episode. The data for training the cluster classifier is randomly sampled from the replay buffer, and the data for generating the MCT^2 is sampled based on the maximum return in each trajectory.

Context Selection: Our framework selects routes from the root to the leaves based on the node values from sample-based probabilistic upper confidence tree (PUCT) Silver et al. (2017). Specifically, we greedily select routes based on the following formula:

$$Sampled\ PUCB = Q(c, a) + c_{ucb} \times \pi(a|o_t) \times \sqrt{(\frac{\log \Gamma_{child}}{1 + \Gamma_{parent}})}. \tag{9}$$

Here, $Q(c, a)$ is the expected discounted cumulative reward gained from the rollout process that starts from that node, c_{ucb} is a hyperparameter for balancing exploration and exploitation, $\pi(a|o_t)$ is the probability of visiting this node, and Γ_{child} and Γ_{parent} represent the number of times this node and its parent node have been visited. After selection, a path of clusters with the largest PUCB values is selected from the root to a leaf. The states and observations stored in the clusters are also acquired to train the contextual predictor.

4.2.2 Contextual Predictor Training

Our framework employs a GRU-based contextual predictor that predicts the next observation $\Phi_{obs}(\hat{o}_{t+1}|o_t, \omega)$ and imitates the actions $\Phi_{act}(\hat{a}_t|o_t, \omega)$ from past experience. Alternatively, other sequence modeling architectures including transformer can also be used for providing contexts. During each training iteration, the data sampled from the MCT^2 is used to supervise the contextual predictor of each agent. Due to the fact that GRU cells are used in the predictor network, the same observations which occur at different time steps among the trajectories are regarded as different nodes. Specifically, the predictor of each agent is conditioned only on the observation o_t^i and the hidden states that the agent acquires. Since observations and joint actions are concatenated in the tree, it is straightforward to extract the corresponding observation and action for each agent.

The training of the contextual predictor is divided into two categories: one-hot action classification and continuous observation regression. The loss function is divided into two parts accordingly. The cross-entropy loss is calculated for action selection and the mean squared error (MSE) loss is used for observation regression:

$$L_{context}(\omega) = \sum_{i=0}^{b} (CE(a_t, \Phi_{act}(\hat{a}_t | o_t, \omega)) + MSE(o_{t+1}, \Phi_{obs}(\hat{o}_{t+1} | o_t, \omega))). \tag{10}$$

Here, CE represents the cross-entropy loss, a_t is the one-hot encoding of the selected action, and $\hat{a}_t$ is the predicted action. MSE denotes the mean squared error, $ot + 1$ is the ground truth next observation, and $\hat{o}_{t+1}$ is the predicted observation. The loss function is minimized using backpropagation and stochastic gradient descent.

4.3 Training Pipeline

In the training process, the utility network is updated by the sum of squared TD error loss, and our contextual predictor is updated according to the CE loss and MSE loss. Therefore, the total loss is combined with these two models' loss:

$$L(\omega) = \sum_{i=0}^{T} L_{utility}(\theta) + \sum_{i=0}^{d_{leaf}} L_{context}(\omega) \tag{11}$$

In the formula above, θ is the network parameters of the utility network which are provided by baseline algorithms and ω is the network parameters of our contextual predictor. In the training process of the utility network, the loss is computed as the sum of errors from time step 0 to the termination step. However, the exploration phase expands only one node in the tree, which may lead to a trajectory that is not terminated. Therefore, the total loss of our contextual predictor is computed as the sum of errors from the root (time step 0) to the leaf node's time step.

During the rollout phase, the supervised trajectory predictor outputs joint actions and predicts the next observations, which are based on learned trajectories. After receiving the observation of time step $t + 1$, the framework compares the true observation o_{t+1} with the predicted observation $\hat{o}_{t+1}$. If the error between them is smaller than a threshold value d_{t+1}, the trajectory predictor continues to make decisions for the next time step. If the error exceeds d_{t+1}, the MARL model takes over the decision-making task from the next time step $t + 1$. We refer to this process as a **switch function** which controls the actions selection

Algorithm 1 The overall algorithm of our framework

```
 1: Warm-up $t_{interval}$ episodes and collect data
 2: Initialize the trajectory pool / tree.
 3: while within the maximum number of rollout time steps do
 4:     while every specific interval time steps do
 5:         Sample trajectories from pool / tree. (Equation 9)
 6:         Train the contextual predictor. (Equation 10)
 7:         Train utility networks by TD loss. (Equation 2)
 8:         Delete trajectories randomly from the pool / destroy and reconstruct the trajectory
            tree. (Subsection 4.2)
 9:     end while
10:     Create a simulator environment and start a new instance.
11:     Set a flag whether utility networks take over the decision-making process: relay=false.

12:     while relay is false do
13:         Obtain predicted action and observation: $\hat{a}_t$, $\hat{o}_{t+1} := predictor(o_t)$
14:         Apply the action and obtain real observation: $o_{t+1} = \text{env.step}(\hat{a}_t)$
15:         relay = check the distances between $o_{t+1}$ and $\hat{o}_{t+1}$ (Equation 12)
16:     end while
17:     Simulate from $\hat{o}_{t+1}$ by utility networks from MARL algorithms.
18:     Calculate discounted cumulated rewards $G^t$
19:     Insert trajectory into pool / Update trajectory tree. (Algorithm 2)
20: end while
```

by observation distances, and its policy is represented as $\pi_{,t} = (\mathbb{1}_t)\pi_{,t}^{Context} + (1 - \mathbb{1}_t)\pi_{,t}^{Util}$, where

$$\mathbb{1}_t = \begin{cases} 1, & |\hat{o}_t - o_t| \leq d_t \text{ and } \mathbb{1}_{t-1} = 1 \\ \\ 0, & \text{others} \end{cases} . \tag{12}$$

In the formula above, $d_t = 1 - (1/e^{d_{end} - d_{\Delta} \cdot t})$, where $d_{\Delta} = (d_{end} - d_{start})/T$.

When the trajectory predictor fails to predict the transition (o_t, a_t, o_{t+1}), the MCT^2 expands according to this transition. The framework selects the node followed by the action sequence $(a_0, ..., a_{t-1})$ and expands it by adding a branch of a_{t+1}, reaching a new node that contains o_{t+1}. Algorithm 1 shows the pseudo-code of our approach.

5. Experiments

We evaluate the performance of our proposed contextual prompting framework via the particle world environment by the average expected reward received by all the agents in an episode. In this environment, we mainly consider three tasks, including Tag, Reference, and Spread tasks. We compare our framework with two value-based MARL algorithms, IQL and QMIX, and one algorithm in single agent scenarios, DRQN. In the following subsections, we will mainly focus on the following aspects: Our framework can make improvements

Algorithm 2 Trajectory tree update algorithm

Input: Transitions along a trajectory with return G_t

```
1: Let r = root of trajectory tree
2: while Each transition $(o_t, a_t, o_{t+1})$ do
3:     if r has no $a_t$ branch then
4:         Expand a new node $o_{t+1}$ with branch $a_t$
5:         return
6:     end if
7:     r = child node with the same action and minimum $dist(o_{t+1}, o_{child,t+1})$
8:     if distance is larger than threshold then
9:         Expand a new node $o_{t+1}$ with branch $a_t$
10:        return
11:    end if
12: end while
13: Back propagation values by discounted $G^t$ along the route. (Equation 8)
```

in convergence speed and final performance compared with both CTDE and non-CTDE MARL algorithms. Ablation studies are also conducted to explain why our framework works by proposing a tree implementation and its pool variant. Additionally, we test our approach on three tasks with different difficulties from SMAC environment to analyze the prompting time steps that the switch function is invoked.

5.1 Experiment Settings

Particle World Environment is a simple multi-agent environment with continuous observation and discrete action space, along with some basic simulated physics. Experiments on this environment are based on the implementation of petting-zoo (Terry et al., 2021) and the settings follow the default parameters. The QMIX and IQL algorithms are provided by pymarl (Samvelyan et al., 2019) which is a framework for deep multi-agent reinforcement learning. This paper mainly focuses on these three tasks:

Spread: N agents cooperate through moving actions to reach N landmarks without collision. Rewards are given by the distances between each agent and their closest landmarks. According to the default settings of the environment, -1 marks will be acquired once two agents collide. N controls the number of agents and landmarks.

Tag: N slow predators cooperate to catch N_p fast prey through the N_b obstacles on the map. Rewards are given by the times that predators catch the prey within a certain time. Once a predator catches prey, 10 marks will be given to all predators. Additionally, we implement a heuristic policy for the prey that the prey will always move in directions far from the predators.

Reference: Original reference environment has 2 agents and 3 landmarks of different colors. Each agent wants to get closer to their target landmark, which is known only by the other agents. Both agents are simultaneous speakers and listeners whose communication channels are provided in the environment state. Based on that, we modify this environment to 5 agents and 7 landmarks. In addition, agents are penalized with -1 marks when a collision occurs and the distance to the closest side for movements outside the window.

During the comparison, we mainly focus on the final performance and the sampling efficiency in the main experiments. In ablation studies with different numbers of agents, we take the time steps until convergence into consideration. The baseline MARL algorithms are QMIX and IQL which are the representations of CTDE and non-CTDE algorithms. To test the compatibility of single-agent scenarios, we compare our prompting DRQN algorithm with DRQN only. Performances are evaluated according to the average expected return received by all the agents with 5 seeds and the shaded areas in the graphs are the variance.

5.2 Experimental Results and Analysis

In this subsection, we will mainly show our experimental results on the three scenarios mentioned above. We conduct experiments on the three tasks and test the learning curve performance within 800k steps. Results of QMIX and IQL only, which represent CTDE and non-CTDE algorithms, and our prompting QMIX and IQL are shown.

Env	timesteps	QMIX(T)	QMIX(P)	QMIX
Spread	5k	**-156.18** $\pm$ 7.56	-157.79 ± 9.44	-160.66 ± 11.22
	50k	**-120.95** $\pm$ 2.18	-154.91 ± 35.93	-143.07 ± 31.76
	300k	-114.60 ± 3.28	-120.04 ± 4.01	-125.88 ± 20.65
	700k	-114.05 ± 2.46	-123.99 ± 6.54	-119.47 ± 19.21
Tag	5k	**40.61** $\pm$ 8.72	11.80 ± 3.05	10.72 ± 2.62
	50k	44.72 ± 9.32	11.85 ± 2.16	12.04 ± 4.49
	300k	**114.94** $\pm$ 57.14	22.50 ± 3.76	20.84 ± 7.13
	700k	**153.83** $\pm$ 20.12	49.52 ± 13.66	56.05 ± 8.03
Reference	5k	**-57.81** $\pm$ 5.73	-108.98 ± 8.71	-84.38 ± 9.92
	50k	**-51.57** $\pm$ 5.56	-70.32 ± 24.68	-60.95 ± 12.90
	300k	-47.55 ± 3.80	-49.80 ± 16.27	-51.95 ± 6.10
	700k	-41.61 ± 1.21	-41.50 ± 3.49	-46.68 ± 6.45
Average	-	**-29.19** $\pm$ 10.59	-60.97 ± 11.81	-57.78 ± 11.71

Table 1: (1/2) Cumulative returns of the two algorithms on four training time steps based on three different environments. The average values and the variances are calculated among 5 different seeds. Among the three environments, higher scores indicate higher performance. Additionally, higher variances represent larger exploration ability during the training process yet lower stability after an algorithm converges.

Table 1 and Table 2 show the cumulative returns of IQL, QMIX, prompting IQL, and prompting QMIX with two variants on the three scenarios. In the table, (T) is an abbreviation of Tree and (P) is short for Pool implementation variants. We also show the results on 5k, 50k, 300k, and 700k time steps, in which 5k and 50k represent early training steps and 700k time step refers to convergence performance. Compared with QMIX and IQL only, in the three environments our proposed prompting QMIX achieves 7 top scores. Our proposed prompting IQL achieves 5 top scores. The average performance is calculated and shows the outperformance of our framework.

Env	timesteps	IQL(T)	IQL(P)	IQL
Spread	5k	-173.11 ± 19.56	-176.69 ± 41.32	-214.48 ± 6.36
	50k	-124.44 ± 2.97	-198.79 ± 46.88	-210.68 ± 48.02
	300k	$\mathbf{-112.89} \pm 2.77$	-189.38 ± 42.42	-166.16 ± 50.91
	700k	$\mathbf{-110.53} \pm 3.43$	-112.14 ± 3.26	-113.05 ± 2.49
Tag	5k	38.81 ± 3.91	11.14 ± 3.43	9.71 ± 3.35
	50k	$\mathbf{46.10} \pm 7.63$	11.29 ± 2.47	12.10 ± 3.09
	300k	79.88 ± 11.84	20.61 ± 6.96	22.50 ± 3.77
	700k	95.38 ± 18.37	28.71 ± 10.18	31.73 ± 10.86
Reference	5k	-66.30 ± 3.97	-102.65 ± 36.72	-101.76 ± 12.10
	50k	-64.38 ± 7.37	-118.67 ± 33.55	-116.06 ± 52.00
	300k	-56.42 ± 5.04	$\mathbf{-42.79} \pm 3.62$	-45.40 ± 2.75
	700k	-51.41 ± 1.42	$\mathbf{-38.45} \pm 3.18$	-50.81 ± 55.75
Average	-	-41.61 ± 7.36	-75.90 ± 19.50	-78.53 ± 20.96

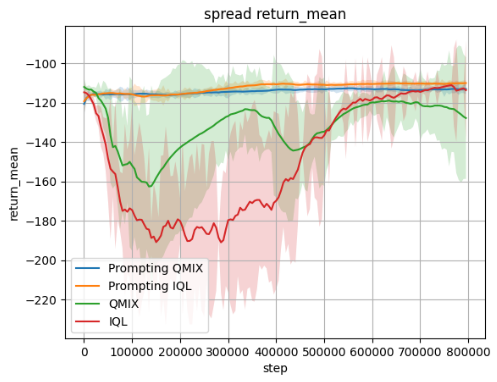
Table 2: (2/2) Cumulative returns of the two algorithms on four training time steps based on three different environments.

Figure 4 shows the learning and testing curves for prompting QMIX (orange), prompting IQL (blue), QMIX(red), and IQL (green) on Spread, Reference, and Tag environments. The x-axis represents the overall execution time steps (800k) and the y-axis is the expected accumulated rewards in an episode.

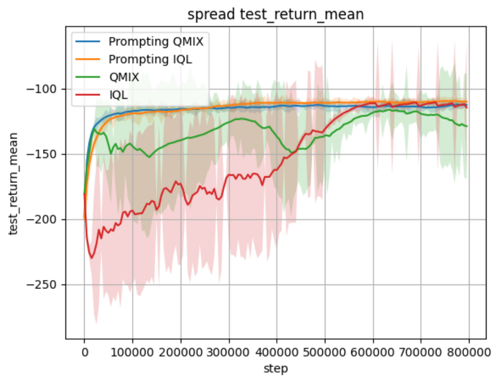
According to the results of Spread and Reference experiments in Figure 4 and convergence proof described above, all the methods with and without contextual prompting reach the final performance of baseline algorithms after their convergence. In the Tag environment, QMIX and IQL methods are on their way to convergence. However, results show that our approach converges much faster than those without prompting. In the Spread environment, our prompting QMIX and prompting IQL with tree implementation spend less than 20k and 50k time steps to converge, but QMIX and IQL need 600k time steps. In the Reference task, our approach takes 80k time steps but QMIX and IQL need about 300k time steps to converge. In the Tag environment, our approach outperforms the baseline algorithms in terms of final performance as well as convergence speed. Additionally, the expected returns of our approach at 5k, 50k, 300k, and 700k are higher than MARL baseline algorithms as shown in Table 1.

The reason for the bad performance of IQL is essentially that agents learn individual policies from their own observations without collaboration. Two agents might collide without cooperation, resulting in reward deduction even though they are on the correct path. QMIX algorithm instead reaches the best performance in most environments, however, the monotonic assumption of QMIX also constrains the performance when minus rewards are given. QMIX is suitable for fully cooperative scenarios instead of competitive scenarios. According to the learning curves of QMIX, the performance drops when a new initial state appears and an increment of agent number will make this phenomenon more severe.

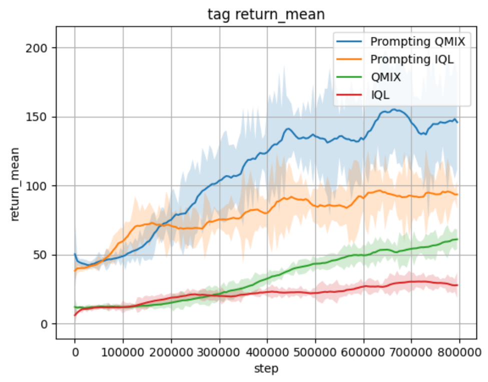
In contrast, the reason for the faster convergence speed of our algorithm is that our supervised predictor generates action according to historical trajectories, which restricts



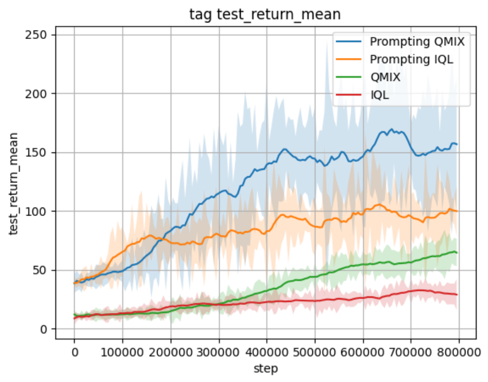
(a) spread training curve



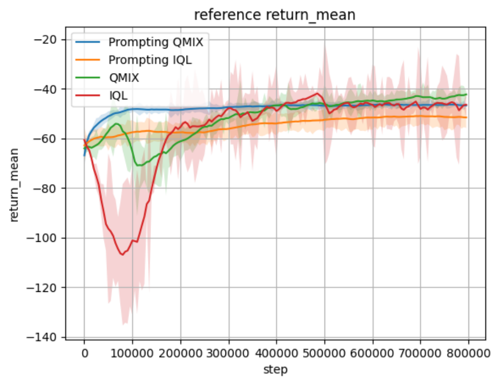
(b) spread testing curve



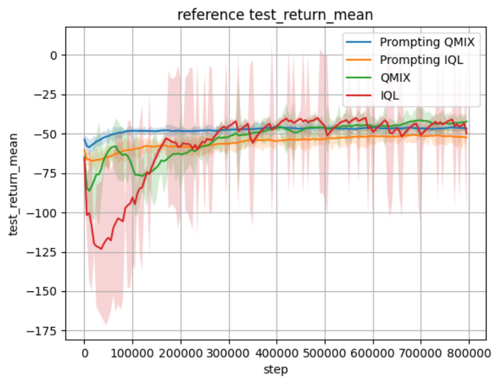
(c) tag training curve



(d) tag testing curve



(e) reference training curve



(f) reference testing curve

Figure 4: Training and testing curves of our framework with QMIX and IQL in the three scenarios.

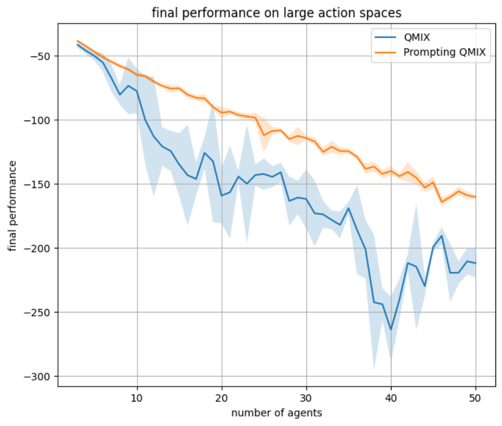
the exploration spaces at early time steps. For most of the partially observable scenarios, actions chosen at early stages have a lower influence on the latter results, and actions chosen at late time steps mostly determine the final result. Therefore, the tree-based supervised model has the ability to quickly eliminate the initial unimportant action sequences. Based on policies learned from these past experiences, policy improvements will be iterated and converge more quickly than purely data-driven algorithms. Additionally, the MCT^2 can act as a centralized training paradigm to overcome the independency of IQL during the training process, such that our prompting IQL method also reaches higher performance.

5.3 Ablation Studies

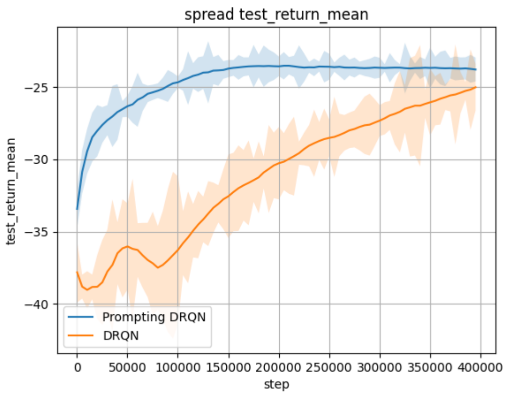
In ablation studies, we conduct experiments on the following three aspects, different scales of action space including single-agent compatibility, two variants of our prompting implementation, and different hyper-parameters of switch function thresholds.

5.3.1 Different Scales on Action Space

We test our approach on large-scale action space scenarios. We conduct our experiment on Spread environment with 3 to 50 agents whose size of action space is from 5^3 to 5^{50} (roughly 10^2 to 10^{35}). We implement QMIX algorithm and our prompting QMIX algorithm for comparison and the testing learning curves of these two algorithms, shown in Figure 5a. Meanwhile, we restrict the maximum number of time steps to 1M so that the training should be terminated once the maximum time step is reached. In the experiment on each number of agents, we conduct our experiment with 5 seeds and the final performance is the average performance among the 5 experiments.



(a)



(b)

Figure 5: (a) Individual final performances on 3-50 agents. (b) The testing curve of our contextual prompting DRQN and DRQN algorithm in Spread scenario with 1 agent.

To test the compatibility of single-agent RL scenarios of our approach, we define the number of agents to 1 in the Spread environment. Then QMIX algorithm for MARL will naturally reduce to the DRQN algorithm for single-agent RL when the mix network is disabled. The learning curves for our prompting DRQN and DRQN are shown in Figure 5b.

According to Figure 5a, as well as Figure 10, 11, 12, 13, we compare our framework with QMIX algorithm with the action space settings from 5^3 (roughly 10^2) to 10^{35}. Results show that our approach is able to achieve acceptable results within 200k time steps and converge within 600k time steps. Additionally, increasing the number of agents or the size of the action space does not affect the final convergence performance. Based on the trend of the convergence curve, predictions can be made that in the tasks with action space larger than 10^{35}, our approach still has the possibility of converging within 1M time steps.

In contrast, the convergence speed for QMIX algorithm is far lower than our framework has. In Figure 10, 11, 12, 13, QMIX algorithm can converge to optimal policies when the action space is small, and when the action space is large, QMIX can only converge to sub-optimal policies in limited time steps. Assumptions can be made that algorithm QMIX will eventually converge to optimal after an enormous number of time steps, so QMIX is still on its convergence way within 1M time steps. As the increment of the action space, QMIX takes more time steps for initial exploration, which slows down the convergence speed. According to the increasing trend of the convergence curve, QMIX algorithm has already possessed the risk of non-convergence when the action space is larger than 10^5.

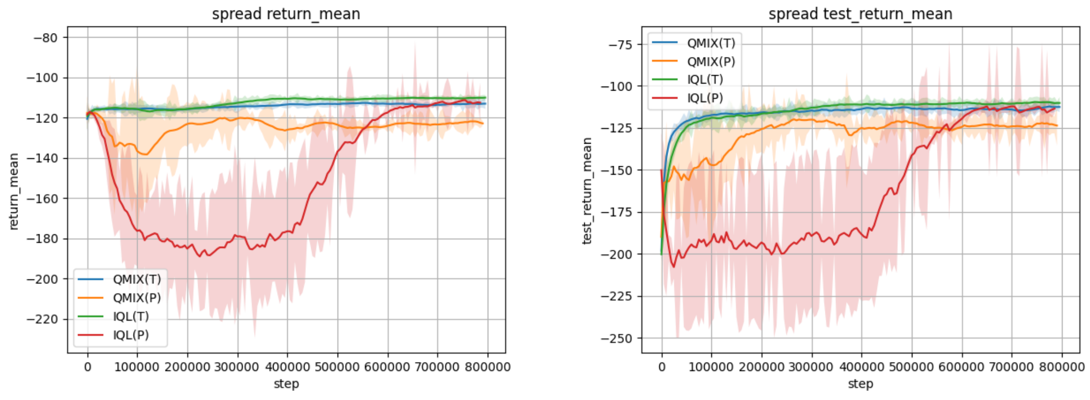
As for continuous action space problems, referring to the previous results, any mechanism that discretizes continuous space can be applied to our framework. To test the compatibility of single-agent RL scenarios of our approach, we define the number of agents to 1 in the Spread environment. Then QMIX algorithm for MARL will naturally reduce to the DRQN algorithm for single-agent RL when the mixing network is disabled. The learning curves for DRQN and prompting DRQN are shown in Figure 5b. Compared to MARL environments, single-agent environments possess much smaller action spaces and more stable environment dynamics, so empirically, our framework will also work in single-agent settings. Results in Figure 5b also show that in single-agent RL settings, our proposed framework can still achieve faster convergence speed with competitive cumulative returns.

5.3.2 Tree vs Pool

To test the functionality of tree implementation, we implement a variant of the trajectory pool. The trajectory pool is the replay buffer that stores data generated from RNN-based algorithms in pymarl codebase. The trajectories for training the contextual predictor are randomly sampled from the replay buffer instead of sampled from the MCT^2. The loss is calculated according to the equation 10 by the randomly selected sequence of transitions. The ablation results are shown in Figure 6

Figure 6 shows the learning curves of the prompting tree and prompting pool implementation, in which the plugin algorithms are QMIX and IQL. The scenario is Spread with 3 agents. T is short for tree and P is short for pool implementation. The x-axis represents interaction time steps and the y-axis stands for the accumulated reward.

As described in the previous section, we implement two trajectory models, trajectory pool and tree in the contextual prompting framework. According to the experimental results in table 1, the average returns of IQL and QMIX with MCT^2 are higher than those with trajectory pool. In the 12 results, QMIX with tree implementation achieves 7 top scores, IQL with tree implementation achieves 3 top scores, and IQL with pool implementation achieves only 2 top scores. Therefore, according to our empirical experimental results,



(a) (b)

Figure 6: Ablation study of tree implementation and pool implementation on promoting QMIX and IQL on Spread environment with 3 agents.

conclusions can be made that the trajectory tree is more proper to be integrated into our framework than the trajectory pool.

Due to the fact that our contextual predictor model classifies the action and predicts the next observation, the optimality of the action in a transition will influence the quality of our predictor model. The trajectories selected from the pool are based on the rank of their returns, however, the high return cannot represent the optimality of the actions in all transitions along that trajectory. Additionally, in one trajectory with high return, agents are probable to choose actions with high instant rewards but low transition possibilities. In this way, the predictor model is trained by these transitions with low accessible probability. The trajectory predictor will always generate high-risk actions and wrong observations.

In contrast, in trajectory tree implementation, trajectories are sampled from the root node to leaf nodes according to greedy PUCB calculation. As opposed to the pool implementation that samples from real trajectories in the replay buffer, our tree implementation samples observations and actions according to the branches and nodes with the largest value. Choosing greedily according to PUCB values guarantees that each transition in one trajectory possesses the best actions. In addition, the Q value of each node is updated during each back-propagation process, which promotes the expectation of the Q value of one node to its true value. Sampling with noise also prevents our predictor model from being overfitted to certain trajectories.

5.3.3 Sequence Modeling Architecture Substitutions

The contextual predictor provides the current best actions directly and predicts the observation in the next time step. This paper applies a GRU-based neural network that is similar to the utility network of each agent. Apart from this, other sequence modeling mechanisms can also be applied such as transformer architecture to provide the action and observation. Thus, we integrate a transformer module and operate an ablation experiment on the use of sequence modeling architectures. The testing curves are shown in Figure 7

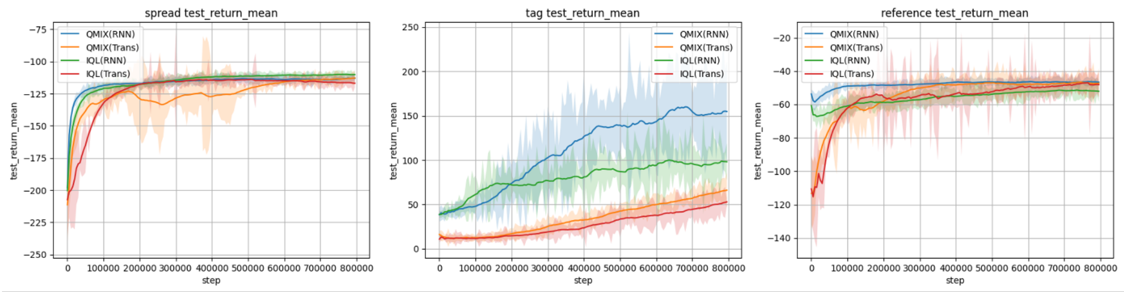


Figure 7: Ablation study of RNN-based contextual predictor and transformer-based contextual predictor on the three scenarios.

In this ablation study, the RNN-based contextual predictor is implemented by GRU cells with MLP and the transformer-based predictor consists of two-layer encoder and decoder and four-head attention blocks. The x-axis is the interactive time steps, and the y-axis is the average return among five seeds. The environment settings are the same as the default settings.

According to the results shown in Figure 7, the RNN-based and transformer-based contextual predictor implementation reaches optimal results within 800k time steps in the Spread and Reference scenarios. In the Tag scenario, RNN-based implementation has higher results than transformer-based implementation. Meanwhile, in each of the scenarios, the convergence speed of transformer-based implementation is about twice lower than that of RNN-based predictors, however, the speed is also much higher than those without promoting modules.

The reason for the slower convergence speed of transformer-based implementation is that the transformer provides strong prediction ability as well as large and complex network size to tune. Compared with training a layer of GRU cells and two layers of fully connected networks, the training time for the transformer is longer. After the transformer training process and the exploration ratio annealing process, the predictor and the utility network begin to take effect. Due to the fact that the restriction on the exploration process is on the tree implementation, whether the contextual predictor is transformer-based or RNN-based does not affect the overall acceleration functionality.

5.3.4 Switch Function Thresholds

The switch function controls which actions, actions from the utility network and actions from contextual predictor, are selected according to the similarity of observation prediction. The similarity is calculated by a cosine distance between the predicted and true observation in the next time step. Therefore, we operate an ablation experiment on the choice of hyper-parameter to find the influence of minimum similarity distance. Figure 8 shows the experimental results.

Figure 8 shows the learning curves of the spread environment with different similarity thresholds at layer 0 of the tree. The scenario is Spread with 3 agents, the x-axis represents the time steps and the y-axis stands for the accumulated return.

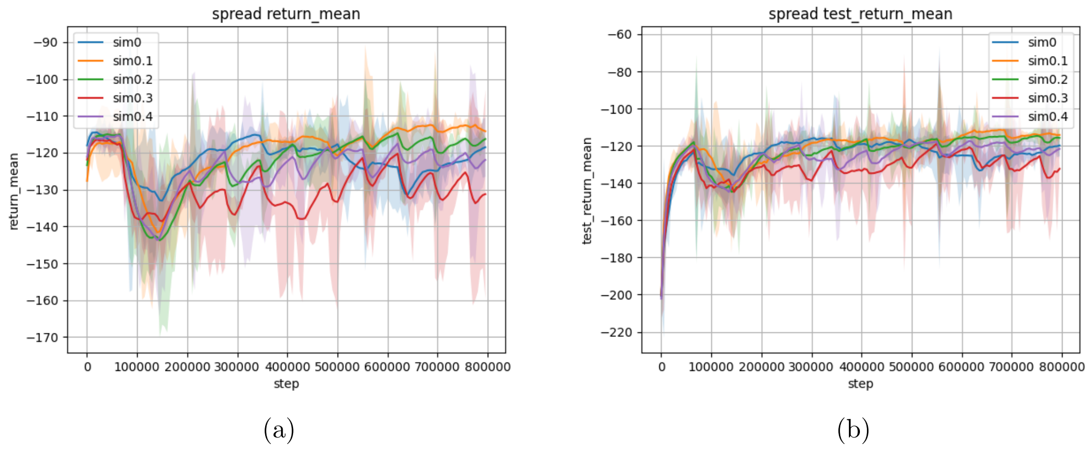


Figure 8: Ablation study of observation similarity threshold in the switch function.

The minimum distance among observations determines the classification of an observation into a cluster and the collaborative MAC. Empirically, a larger distance tolerance restricts the expansion process, such that the next observation predicted by the contextual predictor can be easily grouped into some clusters. In contrast, smaller distances will widen the branches of the trajectory tree because similar observations are regarded as different nodes.

Results in Figure 8 show the learning performance with different minimum similarities. Among the five hyper-parameters settings, the similarity 0.1 achieves the best performance and quickly converges. Compared with the distance of 0.1, the learning curve of similarity 0 is still in the process of convergence, and that of 0.4 reaches suboptimal performance and unstable performance. When the initial state changes in the training process, models with 0.3 also fail to infer optimal actions.

5.3.5 Prompting Time Steps

Apart from the PWE environment, we also validate our prompting algorithms on three sub-tasks with three difficulties on SMAC (Samvelyan et al., 2019) environment which needs enough coordinated explorations. The three scenarios are 1c3s5z (Easy), 3s_vs_5z (Hard), and MMM2 (Super Hard). During the training process, our proposed contextual predictor provides an action and predicts the observation on the subsequent time step. During the rollout process, when the predictor fails to predict the observation correctly, the utility network will take over the decision-making process, which is the relay time step. Therefore, we record the portion of time steps that contextual predictor takes effect with the average length of an episode. We also analyze the influence of the dropout time step on the performance in three scenarios. The experimental results are shown in Figure 9.

According to the task specifications of the three scenarios, 1c3s5z is an easy task that agents need to focus fire on correct enemies, so the contextual predictor has a greater probability of predicting correct observations. In the 3s_vs_5z scenario, in contrast, agents should walk and attack which is difficult for the predictor to predict. The average length

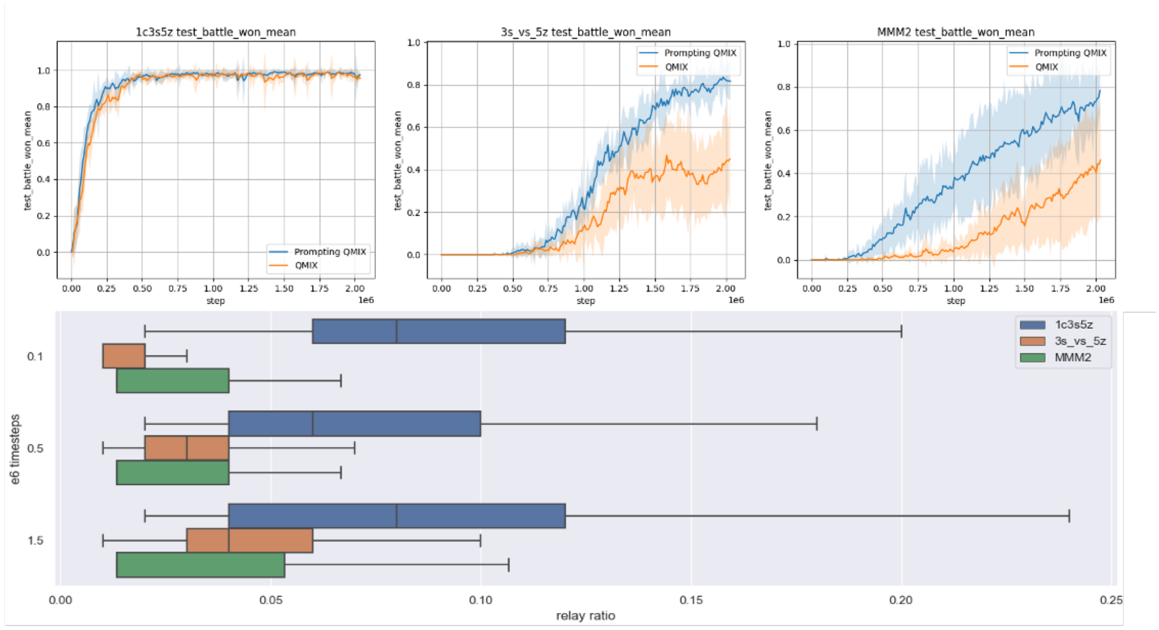


Figure 9: The upper graph shows the learning curves of our prompting QMIX algorithm and QMIX algorithm on three SMAC tasks, 1c3s5z, 3s_vs_5z, and MMM2. The y-axis is the average winning rate among 5 seeds. The lower graph shows the relay time step ratio of our prompting QMIX algorithm on the 1c3s5z, 3s_vs_5z, and MMM2 scenarios in 100k, 500k, and 1500k time steps. The y-axis is the division of the relay time step by the average interactive time steps in an episode.

of an episode of this scenario is 100. The important way to win MMM2 is to control the Medivac to heal the agent and protect itself. Meanwhile, the average length of an episode is 75, so the relay ratio is also high.

Additionally, according to the trend from the graph, as the policy improvement of the utility networks and the improvement in predicting ability of the contextual predictor, the relay ratio becomes higher in later training time steps. Our prompting method restricts the exploration process by prompting network in the early time steps and restores the exploration process when the relay process occurs. Compared with the PWE environment where agents are generated randomly in the map, the starting states of SMAC sub-tasks are quite similar, which makes our prompting algorithms more applicable.

6. Conclusion & Future Work

In this work, we consider the problem of multi-agent system convergence with large-scale action space. To solve the problem, we propose a framework consisting of contextual trajectory tree, context predictor, and baseline MARL algorithms. We show that the usage of trajectories sampled from our trajectory tree on the predictor can accelerate the convergence speed and achieve competitive or outperforming results. Additionally, we justify the optimality of the policy based on sampled action selections and conduct experiments on scenarios with large numbers of agents. Experimental results indicate that our framework can be adapted to value-based MARL methods in terms of implementation and offers significant improvements to value-based MARL methods.

Although our framework has significant improvement based on value-based MARL algorithms, our framework in contrast has limited enhancement on policy-based algorithms. Our work aims to influence the trajectory in the replay buffer so that the deep Q-networks converge faster, which results in overall acceleration. However, in policy gradient algorithms, actions are generated from policy networks, and gradients are back-propagated through the network. The actions from the supervised predictor do not contribute to the policy network updates. In the future, we might consider the predicted actions as counterfactual baselines to improve the actor network.

7. Acknowledgements

This work was supported by the Zhejiang Provincial Natural Science Foundation of China under Grant No. LZ23F020009, the NSFC project (No. 62072399), MoE Engineering Research Center of Digital Library, China Research Centre on Data and Knowledge for Engineering Sciences and Technology, the Fundamental Research Funds for the Central Universities, and Ant Group. We also express our sincere gratitude to anonymous reviewers for their invaluable feedback and constructive comments.

Appendix A. Sample-based Joint-Actions Convergence Justification

(Hubert et al, 2021) gave the following detailed proof[3]:

Lemma 1 *If $\hat{Z}_\beta(s)$ satisfies $\sum_{a\in\mathcal{A}}(\hat{\beta}/\beta)(a|s)f(s,a,\hat{Z}_\beta(s))=1$, it can be obtained that:*

$$\lim_{K\to\infty}\hat{Z}_\beta = Z$$

Proof 1 *It is clear that $\lim_{K\to\infty}\hat{\beta}=\beta$. So we can calculate*

$$\begin{aligned} 1 &= \lim_{K\to\infty}\sum_{a\in\mathcal{A}}(\hat{\beta}/\beta)(a\mid s)f\left(s,a,\hat{Z}_\beta(s)\right) \\ &= \lim_{K\to\infty}\sum_{a\in\mathcal{A}} f\left(s,a,\hat{Z}_\beta(s)\right) \end{aligned} \tag{13}$$

With the uniqueness of Z, it can be proved that $\lim_{K\to\infty}\hat{Z}_\beta = Z$.

Theorem 1 *Given a random variable X,*

$$\mathbb{E}_{a\sim\mathcal{I}\pi}[X\mid s] = \lim_{K\to\infty}\sum_{a\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(a\mid s)X(s,a).$$

Moreover,

$$\sum_{a\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(a\mid s)X(s,a)\sim\mathcal{N}\left(\mathbb{E}_{a\sim\mathcal{I}\pi}[X\mid s],\frac{\sigma^2}{K}\right)$$

where $\sigma^2=\mathrm{Var}_{a\sim\beta}\left[\frac{f(s,a,Z(s))}{\beta}X(s,a)\mid s\right]$.

Proof 2

$$\begin{aligned} &\mathbb{E}_{a\sim\mathcal{I}_\pi}[X(s,a)\mid s] \\ &= \mathbb{E}_{a\sim\beta}[(\mathcal{I}\pi/\beta)(a\mid s)X(s,a)\mid s] \\ &= \mathbb{E}_{a\sim\beta}[f(s,a,Z(s))/\beta(a\mid s)X(s,a)\mid s] \\ &= \lim_{K\to\infty}\sum_{a\in\mathcal{A}}(\hat{\beta}/\beta)(a\mid s)f(s,a,Z(s))X(s,a) \\ &= \lim_{K\to\infty}\sum_{a\in\mathcal{A}}(\hat{\beta}/\beta)(a\mid s)f\left(s,a,\hat{Z}_\beta(s)\right)X(s,a) \\ &= \lim_{K\to\infty}\sum_{a\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(a\mid s)X(s,a) \end{aligned} \tag{14}$$

Based on the above, we can obtain the corollary:

$$\lim_{K\to\infty}\hat{\mathcal{I}}_\beta\pi = \mathcal{I}\pi$$

by using $X(s,a)=\mathbb{1}(a)$[4] , with $\mathcal{I}\pi(a\mid s)=\mathbb{E}_{a\sim\mathcal{I}\pi}[\mathbb{1}(a)\mid s]$ and $\hat{\mathcal{I}}_\beta\pi(a\mid s)=\sum_{b\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(s,b)\mathbb{1}(a)$.

3. In the referenced paper, you can see a more detailed proof.
4. $\mathbb{1}(a)$ means the Indicator function

Appendix B. Convergence speed on large-scale action spaces

We show the rest of the graphs when the number of agents ranges from 3 to 50.

Appendix C. Sampled action size

In the above section, we have proved the guarantee that learning through sampled joint actions will also converge to optimal policies. In this subsection, we conduct experiments in which the number of sampled choices is varied. In the tree-based model, the number of sampled actions is controlled by the maximum number of branches a node can have. Therefore, we choose 10, 15, 20, 25, and 30 branches as candidate hyper-parameters and analyze the influence of sampled amounts. The experiment environment is the Spread task with 3 agents. The full joint-action space of this task is 5^3. Each agent may choose an action from stopping and moving in four directions.

According to Figure 14, the learning curves of different sampled action sizes finally reach optimal performance, which also supports the conclusion that policies generated by sample-based joint action can be projected to optimal policies. Meanwhile, different sizes of samples only contribute to the convergence speed instead of the final performances.

Appendix D. Computational Overhead

Apart from the original MARL algorithms implementation, QMIX for instance, our proposed method needs additional computational overhead on the cluster methods, generating the MCT^2, selecting the nodes with maximum value, and training a contextual predictor network. Compared with the multi-threading rollout process and the time-consuming actions mentioned above, the time consumption on the calculation of observation similarity and judgment of the relay process can be ignored.

Theoretically, the time complexity of clustering methods depends on the specific algorithm used. For example, in this paper, we use the KMeans clustering method and its time complexity is $O(n \times K \times I \times f)$, where n is the number of points and calculated by the multiplication of sampled trajectories from the replay buffer and the sequence length of each trajectory. K is the pre-defined number of clusters. I is the number of EM iterations and is predefined to 100. f is the number of attributes that is the sum of the length of the observation vector per agent. In addition, the time complexity of DBSCAN is $O(n \log n)$, which is much larger than that of KMeans when the number of points becomes large.

As for the MCT^2, the time complexity is similar to MCTS. The establishment of the MCT^2 is the traverse on the observations and reflecting the observations into cluster IDs, so the time complexity is $O(n \times \ K)$, in which n is the total number of points depending on the number of trajectories and the sequence length, K is the time of calculating the nearest neighborhood to assign cluster-ID. The time complexity of sampling trajectories from MCT^2 is $O(c \times k \times d)$, in which c is the maximum number of children a parent node has and in this paper is a hyper-parameter. k is the number of sampled trajectories and d is the depth of the tree, also the length of the trajectories.

In summary, computational overhead depends on the following controllable parameters including the number of trajectories for establishing the MCT^2, the maximum length of a trajectory, the time interval to destroy and reconstruct the MCT^2, the predefined maximum

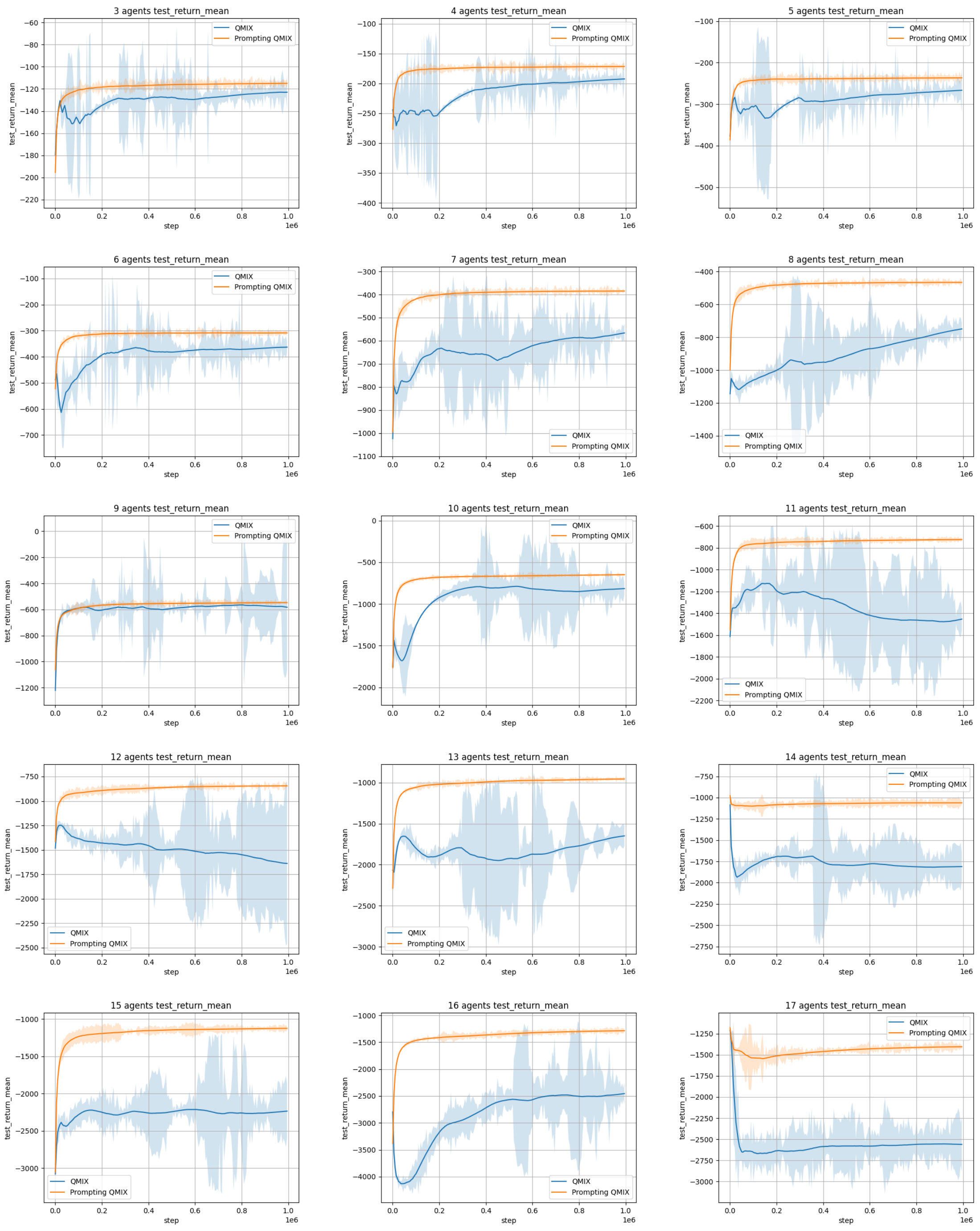


Figure 10: **(1/4)** The learning curves for experiments inside which the agent number is from 3 to 17. Orange lines are our proposed prompting QMIX implementation and the blue lines are QMIX from pymarl implementation. The x-axis is the time steps and the y-axis is the cumulative return of the Spread environment.

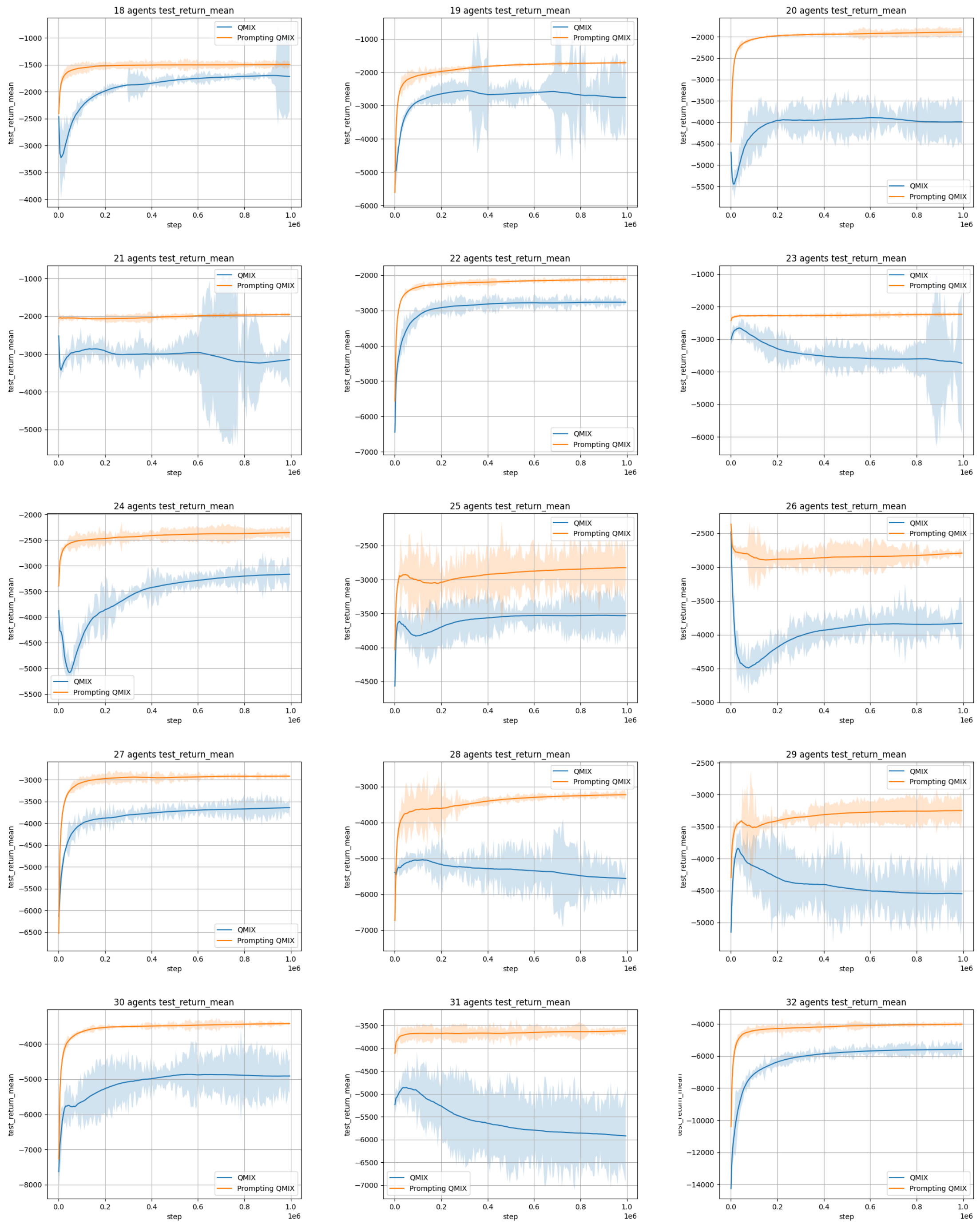


Figure 11: **(2/4)** The continued graphs following Figure 10. The learning curves for experiments inside which the agent number is from 18 to 32. Orange lines are our proposed prompting QMIX implementation and the blue lines are QMIX from pymarl implementation. The x-axis is the time steps and the y-axis is the cumulative return of the Spread environment.

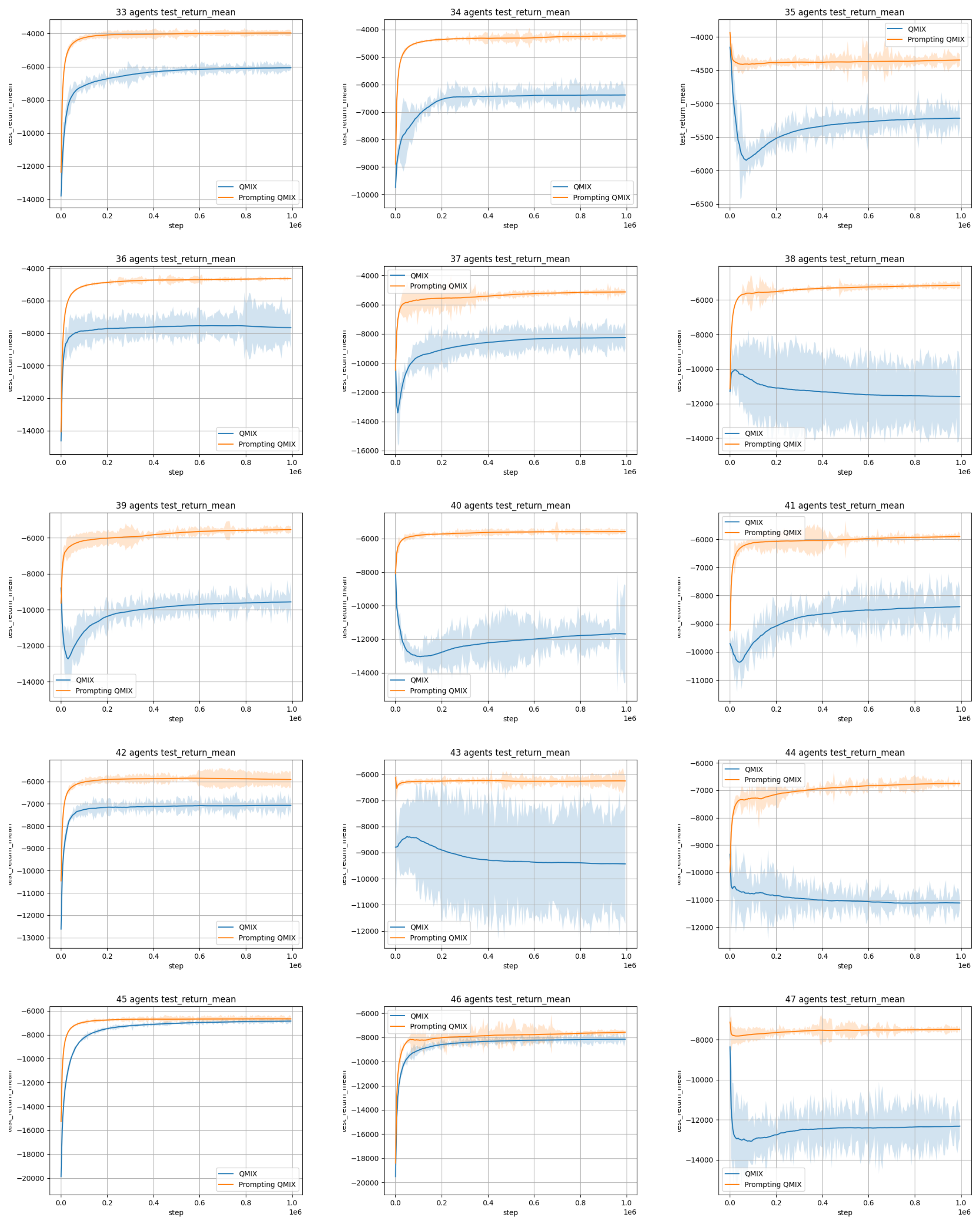


Figure 12: **(3/4)** The continued graphs following Figure 11. The learning curves for experiments inside which the agent number is from 33 to 47. Orange lines are our proposed prompting QMIX implementation and the blue lines are QMIX from pymarl implementation. The x-axis is the time steps and the y-axis is the cumulative return of the Spread environment.

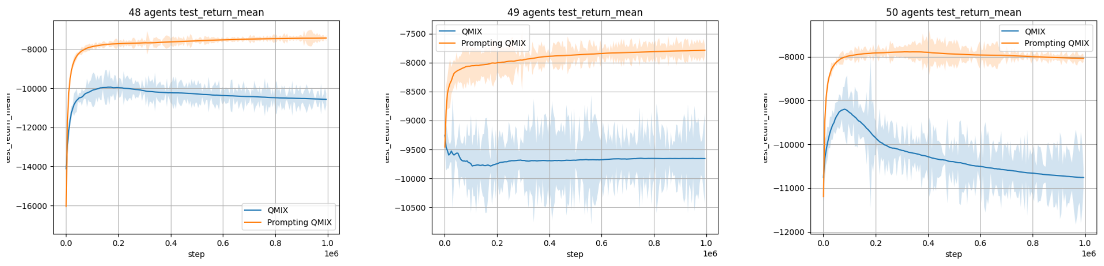


Figure 13: **(4/4)** The continued graphs following Figure 12. The learning curves for experiments inside which the agent number is from 48 to 50. Orange lines are our proposed prompting QMIX implementation and the blue lines are QMIX from pymarl implementation. The x-axis is the time steps and the y-axis is the cumulative return of the Spread environment.

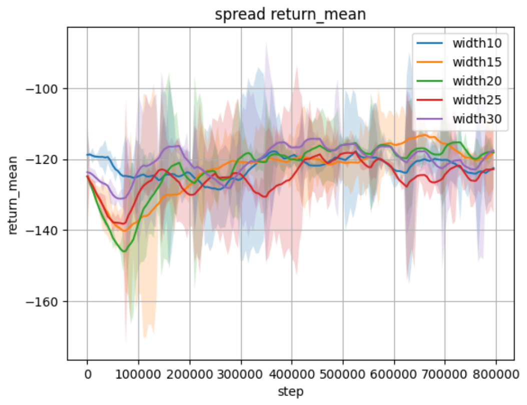


Figure 14: The learning curve of the spread environment with different sample sizes. The experiment hyper-parameters follow the default parameters.

sampled action space, the batch size for training the contextual predictor, and the time interval for training the predictor. Empirically in this paper, with the help of accelerating packages and the choice of hyper-parameters, the overall time used by our framework is twice larger than that of original MARL algorithms. However, our framework is able to converge to the performance of baseline algorithms much faster than the original algorithms.

Appendix E. Hyper-Parameters for Experiments

The experiments are mainly conducted on Spread, Tag, and Reference sub-tasks in particle world environments. Due to the fact that the observation spaces of each agent vary in different sub-tasks and the feature values also represent different meanings with different ranges, calculating the minimum observation similarity is difficult. Meanwhile, the number of agents also affects the similarity distance because we use the concatenation of observations as nodes in the trajectory tree. Therefore, we use different hyper-parameters in each sub-task. The hyper-parameters are shown in Table 3.

Table 3: Hyper-parameters in experiment tasks

Parameters	Spread	Tag	Reference
minimum observation similarity	0.1	0.1	0.1
sample width	15	15	15
prune width	7	7	7
trajectory depth	25	20	25
prune depth	13	10	13
sample interval	400	400	400
λ	1/n	1/n	1/n

Except for the hyper-parameters described in the table, the parameters for algorithm QMIX and IQL as well as those including replay buffer size, learning rates, and the optimizer are the default values provided by pymarl code base. The environment configuration we use is qmix_beta and iql_beta settings. Other hyper-parameters like episode limit are also the default setting introduced by the petting-zoo environment. Note that the minimum observation similarity in the table is the number in detail, this parameter in fact in experiments is dynamically reduced according to the maximum distance from the observation in a cluster with the center observations. Additionally, the minimum observation similarity control is also decayed by the depth of the trajectory tree. Empirically speaking, the similarity control should be tightened as the simulation processes are near the termination goals.

References

Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *Advances in neural information processing systems*, 31, 2018.

Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial informatics*, 9(1):427–438, 2012.

Filippos Christianos, Lukas Schäfer, and Stefano Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. *Advances in neural information processing systems*, 33:10707–10717, 2020.

Filippos Christianos, Georgios Papoudakis, Muhammad A Rahman, and Stefano V Albrecht. Scaling multi-agent reinforcement learning with selective parameter sharing. In *International Conference on Machine Learning*, pages 1989–1998. PMLR, 2021.

Xiangxiang Chu and Hangjun Ye. Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1710.00336*, 2017.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

Simon S Du, Sham M Kakade, Ruosong Wang, and Lin F Yang. Is a good representation sufficient for sample efficient reinforcement learning? *arXiv preprint arXiv:1910.03016*, 2019.

Haotian Fu, Hongyao Tang, Jianye Hao, Chen Chen, Xidong Feng, Dong Li, and Wulong Liu. Towards effective context for meta-reinforcement learning: an approach based on contrastive learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7457–7465, 2021.

Dibya Ghosh, Marlos C Machado, and Nicolas Le Roux. An operator view of policy gradient methods. *Advances in Neural Information Processing Systems*, 33:3397–3406, 2020.

Thomas Gilles, Stefano Sabatini, Dzmitry Tsishkou, Bogdan Stanciulescu, and Fabien Moutarde. Home: Heatmap output for future motion estimation. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 500–507. IEEE, 2021.

Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mohammadamin Barekatain, Simon Schmitt, and David Silver. Learning and planning in complex action spaces. In *International Conference on Machine Learning*, pages 4476–4486. PMLR, 2021.

Vineet Kosaraju, Amir Sadeghian, Roberto Martín-Martín, Ian Reid, Hamid Rezatofighi, and Silvio Savarese. Social-bigat: Multimodal trajectory forecasting using bicycle-gan and graph attention networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

Jakub Grudzien Kuba, Ruiqing Chen, Munning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*, 2021.

Aviral Kumar, Abhishek Gupta, and Sergey Levine. Discor: Corrective feedback in reinforcement learning via distribution correction. *Advances in Neural Information Processing Systems*, 33:18560–18572, 2020.

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pages 5639–5650. PMLR, 2020.

Junwei Liang, Lu Jiang, Juan Carlos Niebles, Alexander G Hauptmann, and Li Fei-Fei. Peeking into the future: Predicting future person activities and locations in videos. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5725–5734, 2019.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*, 2017.

Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. *Advances in Neural Information Processing Systems*, 32, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. *arXiv preprint arXiv:1810.01257*, 2018.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *International Conference on Machine Learning*, pages 3878–3887. PMLR, 2018.

Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Pomdps for robotic tasks with mixed observability. In *Robotics: Science and systems*, volume 5, page 4, 2009.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4292–4301, 2018.

Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 33:10199–10210, 2020.

Christoph Rösmann, Malte Oeljeklaus, Frank Hoffmann, and Torsten Bertram. Online trajectory prediction and planning for social robot navigation. In *2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1255–1260. IEEE, 2017.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

Seyed Kamyar Seyed Ghasemipour, Shixiang Shane Gu, and Richard Zemel. Smile: Scalable meta inverse reinforcement learning through context-conditional policies. *Advances in Neural Information Processing Systems*, 32, 2019.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, pages 9767–9779. PMLR, 2021.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5887–5896. PMLR, 2019.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2085–2087. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

Bohan Tang, Yiqi Zhong, Ulrich Neumann, Gang Wang, Siheng Chen, and Ya Zhang. Collaborative uncertainty in multi-agent trajectory forecasting. *Advances in Neural Information Processing Systems*, 34:6328–6340, 2021.

J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.

Justin K Terry, Nathaniel Grammel, Ananth Hari, Luis Santos, and Benjamin Black. Revisiting parameter sharing in multi-agent deep reinforcement learning. 2020.

Caroline Wang, Ishan Durugkar, Elad Liebman, and Peter Stone. Dm2: Decentralized multi-agent reinforcement learning via distribution matching. 2023.

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062*, 2020a.

Tonghan Wang, Jianhao Wang, Chongyi Zheng, and Chongjie Zhang. Learning nearly decomposable value functions via communication minimization. *arXiv preprint arXiv:1910.05366*, 2019.

Tonghan Wang, Heng Dong, Victor Lesser, and Chongjie Zhang. Roma: Multi-agent reinforcement learning with emergent roles. *arXiv preprint arXiv:2003.08039*, 2020b.

Ke Xiao, Jianyu Zhao, Yunhua He, and Shui Yu. Trajectory prediction of uav in smart city using recurrent neural networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

Feng Xu, Shengyi Jiang, Hao Yin, Zongzhang Zhang, Yang Yu, Ming Li, Dong Li, and Wulong Liu. Enhancing context-based meta-reinforcement learning algorithms via an efficient task encoder (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 15937–15938, 2021.

Yaodong Yang, Ying Wen, Jun Wang, Liheng Chen, Kun Shao, David Mguni, and Weinan Zhang. Multi-agent determinantal q-learning. In *International Conference on Machine Learning*, pages 10757–10766. PMLR, 2020.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

Maosheng Ye, Tongyi Cao, and Qifeng Chen. Tpcn: Temporal point cloud networks for motion forecasting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11318–11327, 2021a.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34: 25476–25488, 2021b.

Chongjie Zhang and Victor Lesser. Coordinated multi-agent reinforcement learning in networked distributed pomdps. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.