

A Distributed Policy Iteration Scheme for Cooperative Multi-Agent Policy Approximation

Thomy Phan
LMU Munich
thomy.phan@ifi.lmu.de

Lenz Belzner
MaibornWolff

Kyrill Schmid
Thomas Gabor
Fabian Ritz
Sebastian Feld
Claudia Linnhoff-Popien
LMU Munich

ABSTRACT

We propose *Stable Emergent Policy (STEP) approximation*, a distributed policy iteration scheme to stably approximate decentralized policies for partially observable and cooperative multi-agent systems. STEP offers a novel training architecture, where function approximation is used to learn from action recommendations of a decentralized planning algorithm. Planning is enabled by exploiting a training simulator, which is assumed to be available during centralized learning, and further enhanced by reintegrating the learned policies. We experimentally evaluate STEP in two challenging and stochastic domains, and compare its performance with state-of-the-art multi-agent reinforcement learning algorithms.

KEYWORDS

multi-agent learning; policy iteration; decentralized planning

1 INTRODUCTION

Many real-world problems can be modeled as cooperative multi-agent systems (MAS) like warehouse commissioning, task allocation, and fleet management [8, 12, 46]. However, decision making in MAS is extremely challenging due to enormous state and joint action spaces, adaptive agent behavior, and state uncertainty [27].

Centralized control does not scale well in large MAS due to the *curse of dimensionality*, where state and joint action spaces grow exponentially with the number of agents [12, 17]. Therefore, decentralized control is recommended, where each agent makes individual decisions, providing better scalability, robustness, and lower costs [8, 12]. Decentralized approaches typically require a *coordination mechanism* like communication, team modeling, or centralized learning to solve joint tasks and to avoid conflicts [5].

Learning decentralized policies with *multi-agent reinforcement learning (MARL)* in cooperative MAS is challenging due to *non-stationarity*, where all agents adapt their behavior concurrently which can lead to uncoordinated policies [5, 6, 12], and *multi-agent credit assignment*, where the joint action of all agents leads to a single global reward which makes the deduction of the individual contributions difficult for adequate adaptation [7, 12, 36, 48].

State-of-the-art approaches to MARL adopt the paradigm of *centralized training and decentralized execution (CTDE)*. Since in many cases, learning can take place in a laboratory or in a simulated environment, global information can be integrated into the training process to learn coordinated policies for decentralized execution in the partially observable world [12, 24, 36, 41]. These approaches are

model-free and rely on the approximation of a joint action value function. While working well for a small number of agents, they can converge to poor local optima in larger MAS since the joint action value function can become a performance bottleneck due to the exponentially scaling input space of joint actions.

Recent approaches to learning strong policies are based on *policy iteration* and combine planning with deep reinforcement learning, where a neural network is used to imitate the action recommendations of a tree search algorithm. In return, the neural network provides an action selection prior for the tree search [1, 39]. This iterative procedure, called *Expert Iteration (ExIt)*, gradually improves the performance of the tree search and the neural network [1]. ExIt has been successfully applied to fully observable zero-sum games, where a single agent improves itself by self-play.

While planning is able to recommend higher quality actions than model-free RL [1, 14, 38, 39], policy improvement via planning actually is a computational bottleneck due to the extra simulations required to evaluate each action. Thus in practice, planning is only suitable for training fast policies via function approximation [1, 14]. Applying ExIt directly to large cooperative MAS with a centralized tree search would be practically infeasible due to the intractable search space - even if global information was available [8, 12, 17]. However, decentralized planning could lower the training cost, since dividing the search space and computation among several simple machines is cheaper, offering higher efficiency due to parallelization, and robustness against failures, in contrast to using a single large machine with high computational power [8].

In this paper, we propose *Stable Emergent Policy (STEP) approximation*, a distributed policy iteration scheme to stably approximate decentralized policies for cooperative MAS. Our contributions are as follows:

- We extend ExIt to cooperative and partially observable MAS by proposing a novel training architecture and using function approximation to learn decentralized policies from local action recommendations of a decentralized planner.
- We propose a decentralized planning scheme for scalable policy improvement. To recommend coordinated local actions, we reintegrate the approximated local policies into each planner as action selection prior and as prediction of other agents' behavior. While planning is performed on the fully observable problem formulation, the learned decentralized policies are only conditioned on their local histories, thus being executable in the partially observable world.

- In contrast to state-of-the-art MARL, we additionally exploit the training simulator required for centralized learning [12, 24, 36, 41] as a generative model for Monte Carlo planning to recommend actions for local policy approximation.
- For domains that are too complex to provide sufficient computation budget for closed-loop planning, which conditions on states, we propose decentralized open-loop planning as a scalable instantiation of our decentralized planning scheme.
- We experimentally evaluate the key elements of STEP in two challenging and partially observable domains and show that STEP is able to learn stronger policies than state-of-the-art MARL approaches w.r.t. the number of simulated time steps and when the number of agents is large.

2 BACKGROUND

2.1 Problem Formulation

Single-agent problems can be formulated as *Markov Decision Process (MDP)* $M_{MDP} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where \mathcal{S} is a set of states s_t , \mathcal{A} is the set of actions a_t , $\mathcal{P}(s_{t+1}|s_t, a_t)$ is the transition probability, and $r_t = \mathcal{R}(s_t, a_t)$ is a scalar reward. MDPs can be extended to MAS by formulating a *stochastic game (SG)* M_{SG} , where \mathcal{S} , \mathcal{P} , and \mathcal{R} remain the same. In addition, M_{SG} defines $\mathcal{D} = \{1, \dots, N\}$ as a set of agents, $\mathcal{A} = \mathcal{A}_1 \times \dots \times \mathcal{A}_N$ as the set of *joint actions* a_t , \mathcal{Z} as a set of local observations $z_{t,i}$ for each agent $i \in \mathcal{D}$, and $\Omega(s_t, a_t) = z_{t+1} = \langle z_{t+1,1}, \dots, z_{t+1,N} \rangle \in \mathcal{Z}^N$ as the joint observation function. Each agent i maintains a local history $\tau_{t,i} = [a_{1,i}, z_{2,i}, \dots, a_{t-1,i}, z_{t,i}]$ with $\tau_t = \langle \tau_{t,1}, \dots, \tau_{t,N} \rangle$ being the *joint history*. Since we focus on cooperative MAS, all agents observe a *global reward* $\mathcal{R}(s_t, a_t)$.

The goal in $M = M_{SG}$ (for brevity we always use M in the following) is to find an optimal *joint policy* $\pi(\tau_t) = \langle \pi_1(\tau_{t,1}), \dots, \pi_N(\tau_{t,N}) \rangle \in \mathcal{A}$, with π_i being the *local policy* or *decentralized policy* of agent i , which maximizes the expectation of return G_t :

$$G_t = \sum_{k=0}^{h-1} \gamma^k \mathcal{R}(s_{t+k}, a_{t+k}) \quad (1)$$

where $\gamma \in [0, 1)$ is the discount factor and h is the horizon.

A joint policy π can be evaluated with a *value function* $V^\pi(s_t) = \mathbb{E}_\pi[G_t|s_t]$, which is the expected return at s_t . π is optimal, if $V^\pi(s_t) \geq V^{\pi'}(s_t)$ for all $s_t \in \mathcal{S}$ and all other joint policies π' . We denote an optimal joint policy by $\pi^* = \langle \pi_1^*, \dots, \pi_N^* \rangle$ and the optimal value function by $V^{\pi^*} = V^*$.

2.2 Planning in MDPs

Planning searches for an (near-)optimal policy, given a model \hat{M} (we use \hat{M} as model for M_{MDP} and M) of the environment M_{MDP} . *Policy iteration* is a *global planning* approach which computes π^* with alternating *policy evaluation*, where V^{π^n} is computed for the current policy π^n , and *policy improvement*, where a *stronger policy* π^{n+1} is generated by selecting actions that maximize V^{π^n} for each state $s_t \in \mathcal{S}$ [18]. *Local planning* only regards the current state s_t and possible future states to find a policy π_t [47]. *Monte Carlo planning* uses a *generative model* \hat{M} as black box simulator without reasoning about explicit probability distributions (e.g., \mathcal{P}) [20, 47].

Monte Carlo Tree Search (MCTS) is a popular local Monte Carlo algorithm, which constructs a search tree to find π_t^* for s_t [20]. The tree is traversed by selecting nodes $s_t \in \mathcal{S}$ with a policy π_{tree}

until a leaf node s_{t+k-1} is reached. UCB1 is a common choice for π_{tree} by maximizing $UCB1 = \overline{Q}(s_t, a_t) + c\sqrt{2\log(n_t)/n_{a_t}}$, where $\overline{Q}(s_t, a_t)$ is the average return, n_t is the visit count of s_t , n_{a_t} is the selection count of a_t , and c is an exploration constant [2, 20]. MCTS using UCB1 is called *UCB1 applied to Trees (UCT)*. The node s_{t+k-1} is expanded by a new child node s_{t+k} , whose value $\hat{V}(s_{t+k})$ is estimated with a rollout or a value function [20, 39, 40]. The observed rewards are recursively accumulated (Eq. 1) to update $\overline{Q}(s_t, a_t)$ of each $\langle s_t, a_t \rangle$ -pair in the search path. MCTS is an *anytime algorithm*, which returns an action recommendation $a_t = \operatorname{argmax}_{a_t \in \mathcal{A}}(\overline{Q}(s_t, a_t))$ for the root state s_t after a *computation budget* n_b has run out. MCTS can be implemented as *closed-loop* or *open-loop* search as explained in [22, 33, 34].

2.3 Reinforcement Learning in MDPs

Reinforcement Learning (RL) searches for an (near-)optimal policy in an unknown environment M_{MDP} without knowing the effect of executing $a_t \in \mathcal{A}$ in $s_t \in \mathcal{S}$ [5, 43]. RL agents obtain experience samples $E = \{e_1, \dots, e_t\}$ with $e_t = \langle s_t, a_t, s_{t+1}, r_t \rangle$ by interacting with M_{MDP} . In *model-free RL*, π^* and/or V^* are directly approximated from E without approximating \mathcal{P} and \mathcal{R} of M_{MDP} [5, 43].

2.4 Decision Making in MAS

Cooperative MAS can be formulated as a joint action MDP and solved with single-agent planning or RL (Section 2.2 & 2.3) by searching for joint policies [5]. However, this does not scale well due to the curse of dimensionality, where the state and joint action spaces grow exponentially with the number of agents [8, 12, 17].

Alternatively, local policies can be searched with decentralized planning or RL, where each agent plans or learns independently of each other [8, 45]. Decentralized approaches require a *coordination mechanism* to solve joint tasks and to avoid conflicts [5]. Common mechanisms are *communication* to exchange information [10, 49], *synchronization* to reach a consensus [9, 28], or *prediction* of other agents' behavior with policy models [8, 35].

3 RELATED WORK

Previous work on policy iteration in MAS has focused on centralized offline planning, where an (near-)optimal joint policy is searched by exhaustively evaluating and updating all local policy candidates for each agent with an explicit model of the MAS [3, 16, 37]. These approaches do not scale well for complex domains due to the curse of dimensionality of the joint policy space. We propose to use *decentralized planning*, where each agent's policy is improved with an independent local search about its *individual actions* with appropriate coordination mechanisms. Furthermore, we do not rely on explicit models but use a *generative model* as black box simulator.

Alternative approaches to centralized planning and learning explicitly exploit structure of the underlying MDP to perform on a factored and more tractable formulation of the problem [13, 21]. We propose a *black box approach* which does not require prior knowledge about the problem structure.

Model-free MARL is a widely studied alternative to policy iteration in MAS, where standard RL algorithms are often performed by all agents independently [23, 44, 45]. While independent learning benefits from scalability, non-stationarity and lacking multi-agent

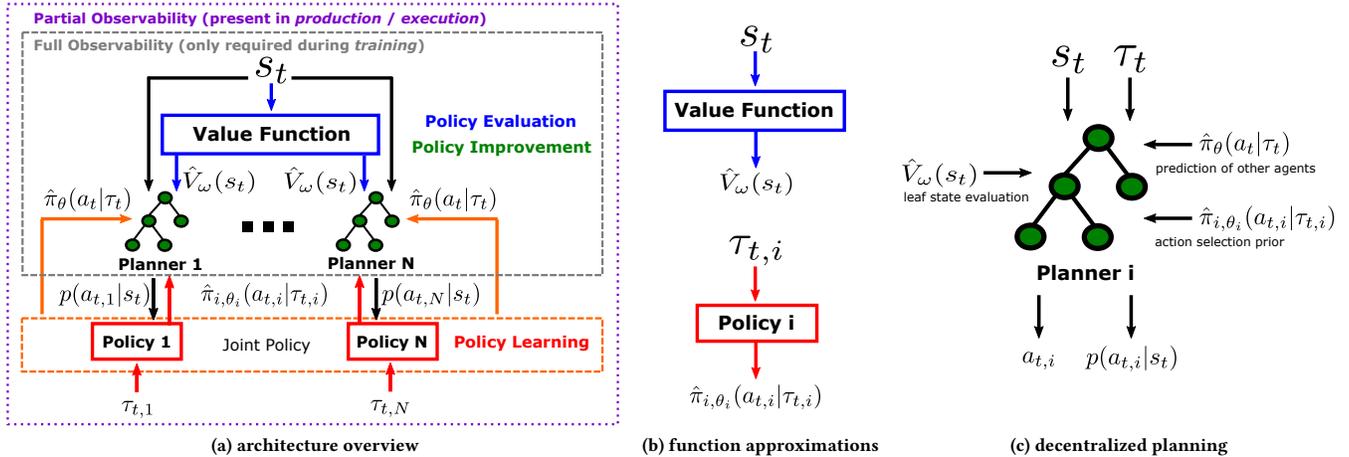


Figure 1: Training architecture of STEP. (a) Information flow between the learned policies $\hat{\pi}_{i,\theta_i}$ (red), the learned value function \hat{V}_ω (blue) and the decentralized planners (green). The policy iteration components (blue and green) within the gray dashed rectangle use global information and are only required during centralized learning, while the learned policies (red) can act under partial observability. (b) and (c) show the black box models of $\hat{\pi}_{i,\theta_i}$, \hat{V}_ω , and the decentralized planners respectively.

credit assignment could lead to uncoordinated policies [5, 6, 12]. Decentralized approaches with adaptive learning rates and opponent modeling techniques can address non-stationarity [11, 25, 28–30] but do not explicitly consider multi-agent credit assignment.

Current state-of-the-art MARL approaches are based on CTDE to address both non-stationarity and multi-agent credit assignment. Since in many cases, learning can take place in a laboratory or in a simulated environment, global information can be integrated into the training process to learn coordinated policies for decentralized execution in the partially observable world. In all cases, a joint action value function is learned to either provide a counterfactual baseline for policy gradient learning [12] or to learn a factorization of that joint action value function [36, 41, 42]. Since these approaches use gradient-based learning and the input space of the joint action value function scales exponentially w.r.t. the number of agents, they can converge to poor local optima in large MAS. We extend the centralized learning concept with decentralized planning by exploiting the *training simulator*, which is assumed in the literature [11, 12, 24, 36, 41] anyway. This enables a decentralized *local action search* on the fully observable problem formulation to escape poor local optima and mitigates the credit assignment problem via explicit reasoning about individual actions. Decentralized policies are learned by *imitating* the action recommendations of the planner and can be executed in the partially observable world without additional planning or global information.

Recently, ExIt-based approaches to learning strong policies from MCTS recommendations with deep learning have been applied to single-agent domains and zero-sum games [1, 14, 38, 39]. In zero-sum games, a single agent is trained via self-play, which corresponds to the policy iteration scheme, where self-play evaluates the current policy and MCTS improves the policy by recommending stronger actions based on the evaluation. Our work extends this approach to *partially observable* and *cooperative multi-agent*

domains, where agents have to achieve a *common goal* by solving joint tasks and avoiding conflicts. Our setting is not turn-based like [1, 38, 39], thus requires *simultaneous planning* of each agent. We additionally propose to use the learned local policies as *coordination mechanism* e.g., to predict other agents’ behavior during decentralized planning.

Ideas of ExIt have been applied to fully observable MAS with a small number of partly fixed agents [19, 35]. In such simple settings coordination is trivial and a single centralized function approximator can be used to approximate both the policy and the value function since they only condition on the global state. We address large *partially observable* multi-agent domains with *multiple learning* agents and propose a novel training architecture in which we exploit the *training simulator* for *decentralized planning* on the fully observable problem to stably learn decentralized policies which can be executed in the partially observable world afterwards.

4 STEP

4.1 Distributed Policy Iteration Scheme

Policy iteration for MAS consists of an alternating *evaluation* and *improvement* step [5]. Given a joint policy $\pi^n = \langle \pi_1^n, \dots, \pi_N^n \rangle$, the global value function V^{π^n} can be computed to evaluate π^n . By selecting joint actions $a_t = \langle a_{t,1}, \dots, a_{t,N} \rangle \in \mathcal{A}$ which maximize V^{π^n} for each joint history τ_t , we obtain an improved joint policy $\pi^{n+1}(\tau_t) = a_t$, which is stronger than π^n . In practice, the number of joint histories and states, and the joint action space are typically too large to exactly compute V^{π^n} and π^{n+1} [5, 8]. Thus, we use function approximation to compute $\hat{V} \approx V^{\pi^n}$ and $\hat{\pi} = \langle \hat{\pi}_1, \dots, \hat{\pi}_N \rangle \approx \pi^{n+1}$.

For *scalable policy evaluation* (Fig. 1a, blue component), we use temporal difference (TD) learning to train \hat{V} with experience and ensure generalization to avoid computing $V^{\pi^n}(s_t)$ for each state $s_t \in \mathcal{S}$ explicitly [43]. Unlike state-of-the-art MARL approaches

[12, 24, 36, 41], we approximate the state value function, which is independent of the joint action dimension (Fig. 1b), thus offering a scalable function representation w.r.t. the number of agents.

For *scalable policy improvement* (Fig. 1a, green components), we use decentralized planning, where each agent i explicitly reasons about the global effect of its individual actions $a_{t,i} \in \mathcal{A}_i$ in state s_t instead of searching the whole joint action space \mathcal{A} . Furthermore, we exploit the simulated training environment assumed in [12, 24, 36, 41] as a generative model to enable decentralized Monte Carlo planning. These proposals alleviate the scalability problem of traditional multi-agent policy iteration approaches, where planning is completely centralized and requires explicit probability distributions of the environment [3, 16, 37]. To further improve planning, we provide the learned value function \hat{V} and the local policy approximations $\hat{\pi}_i$ (see next paragraph) of all (other) agents i .

In addition to policy evaluation and policy improvement, which assume full observability since both value function and decentralized planners condition on the global state s_t (Fig. 1b-1c), we propose a *policy learning* scheme (Fig. 1a, red components), where function approximators $\hat{\pi}_i$ are used to imitate the action recommendations of each agent i 's local planner. Thus, $\hat{\pi}_i$ improves along with its local planner during training. Since $\hat{\pi}_i$ only conditions on the corresponding local history $\tau_{t,i}$ (Fig. 1b), the learned policies can be executed in the partially observable world without additional planning, or global information.

The explicit reasoning via decentralized planning mitigates the multi-agent credit assignment problem, because each agent i is incentivized to optimize its *individual* actions to maximize the *common* return based on the common global value function \hat{V} and the local policy approximations $\hat{\pi}_i$ of all (other) agents i (Fig. 1c). Since $\hat{\pi}_i$ is trained to imitate the individual planner of agent i , $\hat{\pi}_i$ can be used to predict future actions of agent i , to optimize local decisions. This can lead to coordinated actions to solve joint tasks and to avoid conflicts [8].

Combining these elements leads to a *distributed policy iteration* scheme for practically feasible approximation of cooperative multi-agent policies. The only requirement is a generative model which we assume to be available as training simulator anyway according to the CTDE paradigm for state-of-the-art MARL [11, 12, 24, 36, 41].

4.2 Decentralized Open-Loop Planning

We formulate an open-loop variant of MCTS, called *Decentralized Open-Loop UCT (DOLUCT)* as our reference algorithm for decentralized planning. Open-loop planning generally converges to sub-optimal solutions, but it is competitive to closed-loop planning in practice, if the problem is too large to provide sufficient computation budget, due to searching a much smaller space with a branching factor of $|\mathcal{A}_i|$ [22, 31, 33, 34, 47]. In stochastic multi-agent domains, closed-loop planning has a worst case branching factor of $|\mathcal{A}_i||\mathcal{S}|$ which would require an enormous computation budget n_b if the state space \mathcal{S} is large. Fig. 2 provides an example to illustrate the branching factor's dependency of the state space and the environment's stochasticity for open- and closed-loop planning.

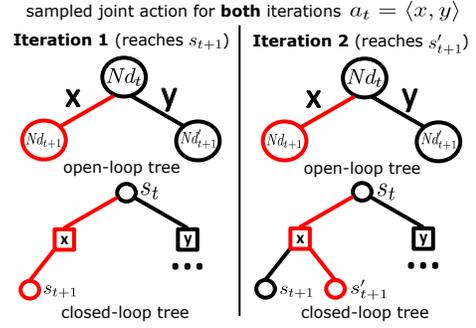


Figure 2: Decentralized planning example for a stochastic two-agent domain with $\mathcal{A}_1 = \mathcal{A}_2 = \{x, y\}$ and $h = 1$ for two simulated iterations from the perspective of agent 1. Even if the same joint action $a_t = \langle x, y \rangle$ is sampled for the root node in both iterations, the environment's stochasticity could lead to different successor states s_{t+1} or s'_{t+1} . While the branching factor of the open-loop tree is independent of the state space, the closed-loop tree has to expand when encountering new states, thus increasing its branching factor.

Algorithm 1 Decentralized Open-Loop UCT (DOLUCT)

```

1: procedure DOLUCT( $i, s_t, \tau_t, \hat{M}, N, n_b, \hat{V}, \hat{\pi}$ )
2:    $\langle \overline{Q}(Nd_t, a_{t,i}), n_{t,i}, n_{a_{t,i}} \rangle \leftarrow \langle 0, 0, 0 \rangle, \forall a_{t,i} \in \mathcal{A}_i$ 
3:   while  $n_b > 0$  do
4:      $\langle G_t, n_b \rangle \leftarrow \text{Simulate}(i, Nd_t, s_t, \tau_t, \hat{M}, N, \hat{V}, \hat{\pi}, n_b)$ 
5:      $p(a_{t,i}|s_t) \leftarrow \frac{n_{a_{t,i}}}{n_{t,i}}, \forall a_{t,i} \in \mathcal{A}_i$ 
6:   return  $\langle \text{argmax}_{a_{t,i} \in \mathcal{A}_i} \overline{Q}(Nd_t, a_{t,i}), p(a_{t,i}|s_t) \rangle$ 

1: procedure Simulate( $i, Nd_t, s_t, \tau_t, \hat{M}, N, \hat{V}, \hat{\pi}, n_b$ )
2:   if  $n_b = 0$  then return  $\langle \hat{V}(s_t), 0 \rangle$ 
3:    $n_b \leftarrow n_b - 1$ 
4:   if  $Nd_t$  is a leaf node then
5:     Expand  $Nd_t$   $\triangleright$  Initialize child node variables with 0
6:     return  $\langle \hat{V}(s_t), n_b \rangle$ 
7:    $a_{t,j} \sim \hat{\pi}_j(a_{t,j}|\tau_{t,j}), \forall j \in \mathcal{D}$   $\triangleright$  predict other agents' actions
8:    $a_{t,i} \leftarrow \text{argmax}_{a_{t,i} \in \mathcal{A}_i} (\text{UCB1}_{Nd_t}^{\hat{\pi}_i}(\tau_{t,i}, a_{t,i}))$   $\triangleright$  Eq. 2
9:    $a_t \leftarrow \langle a_{t,1}, \dots, a_{t,N} \rangle$ 
10:   $\langle s_{t+1}, r_t, z_{t+1} \rangle \sim \hat{M}(s_t, a_t)$   $\triangleright$  simulate joint action
11:   $\langle n_{t,i}, n_{a_{t,i}} \rangle \leftarrow \langle n_{t,i} + 1, n_{a_{t,i}} + 1 \rangle$   $\triangleright$  update  $Nd_t$ 
12:   $\langle R_t, n_b \rangle \leftarrow \text{Simulate}(i, Nd_{t+1}, s_{t+1}, \tau_{t+1}, \hat{M}, N, \hat{V}, \hat{\pi}, n_b)$ 
13:   $G_t \leftarrow r_t + \gamma R_t$ 
14:   $\overline{Q}(Nd_t, a_{t,i}) \leftarrow ((n_{t,i} - 1)\overline{Q}(Nd_t, a_{t,i}) + G_t) / n_{t,i}$ 
15:  return  $\langle G_t, n_b \rangle$ 

```

At every training time step t , all agents perform an independent DOLUCT search in parallel¹. A stochastic joint policy function $\hat{\pi} = \langle \hat{\pi}_1, \dots, \hat{\pi}_N \rangle$ with $\hat{\pi}_i(a_{t,i}|\tau_{t,i}) \in [0, 1]$ for agent $i \in \mathcal{D}$ can be provided to simulate all other agents. To traverse a DOLUCT search

¹Although MCTS is commonly used for online planning, we use MCTS in training simulations for the policy improvement step (Fig. 1a). While MCTS is performed after each time step in the training simulation, the whole training phase itself is *offline*.

tree, we propose a modified version of UCB1 similarly to [39]:

$$UCB1_{Nd_t}^{\hat{\pi}_i}(\tau_{t,i}, a_{t,i}) = \overline{Q(Nd_t, a_{t,i})} + \hat{\pi}_i(a_{t,i}|\tau_{t,i})c\sqrt{\frac{2\log(n_{t,i})}{n_{a_{t,i}}}} \quad (2)$$

where Nd_t is a node in the open-loop tree (Fig. 2) and $\overline{Q(Nd_t, a_{t,i})}$ is the average return of *all visited states* in node Nd_t (Fig. 2). Note that the probabilities $\hat{\pi}_i(a_{t,i}|\tau_{t,i})$ for the same node Nd_t can vary depending on the simulated state s_t and the resulting history $\tau_{t,i}$, thus providing a *closed-loop prior* for the action selection.

To avoid a full depth search, a value function \hat{V} is used to evaluate states at leaf nodes [32, 39].

DOLUCT is formulated in Algorithm 1, where i identifies the agent, s_t is the current state, τ_t is the current joint history, \hat{M} is the generative model, N is the number of agents, n_b is the computation budget, \hat{V} is a value function, and $\hat{\pi}$ is a joint policy.

Note that since there are several approaches to decentralized Monte Carlo planning which only differ in their coordination mechanisms (e.g., communication [4, 32], prediction of other agent’s behavior [8, 35], etc.), we do not consider DOLUCT a major contribution but merely a reference algorithm to demonstrate how our approximated policies can be integrated into the planning process (Section 4.3). Our experiments in Section 6.1 show that our approach is also able to enhance decentralized closed-loop planning.

4.3 Stable Emergent Policy Approximation

With *Stable Emergent Policy (STEP) approximation*, π^* is learned by imitating a decentralized planner similarly to ExIt [1, 39] for single agents. STEP consists of an alternating *planning* and *learning* step.

In the (decentralized) planning step, a joint action $a_t = \langle a_{t,1}, \dots, a_{t,N} \rangle$ is searched (e.g., with DOLUCT) for the current state s_t . The planning algorithm can exploit the local policy $\hat{\pi}_i$ as a prior for action selection (e.g., Eq. 2) and $\hat{\pi} = \langle \hat{\pi}_1, \dots, \hat{\pi}_N \rangle \approx \pi^*$ as prediction of other agents’ behavior for coordination to improve itself (Fig. 1c). All agents execute a_t and cause a state transition to s_{t+1} , while observing a global reward r_t and z_{t+1} . The transition $e_t = \langle s_t, a_t, s_{t+1}, r_t, z_{t+1}, p_t \rangle$ is stored as *experience sample* in a *central buffer* E , where $p_t = \langle p(a_{t,1}|s_t), \dots, p(a_{t,N}|s_t) \rangle$ contains the relative frequencies of the action selections $p(a_{t,i}|s_t) = \frac{n_{a_{t,i}}}{n_{t,i}}, \forall a_{t,i} \in \mathcal{A}_i$ of each agent’s individual planner for state s_t (e.g., Algorithm 1).

Algorithm 2 Stable Emergent Policy (STEP) approximation

```

1: procedure STEP( $\Psi, \hat{M}, N, n_b, \hat{\pi}_\theta, \hat{V}_\omega$ )
2:   Initialize  $\theta$  for  $\hat{\pi}_\theta$  and  $\omega$  for  $\hat{V}_\omega$ 
3:   Observe  $s_1$ 
4:   for  $t = 1, T$  do                                 $\triangleright T$  is the training duration
5:     for  $i \in \mathcal{D}$  do                                 $\triangleright$  decentralized (parallel) planning
6:        $\langle a_{t,i}, p(a_{t,i}|s_t) \rangle \leftarrow \Psi(i, s_t, \tau_t, \hat{M}, N, n_b, \hat{V}_\omega, \hat{\pi}_\theta)$ 
7:      $p_t \leftarrow \langle p(a_{t,1}|s_t), \dots, p(a_{t,N}|s_t) \rangle$ 
8:     Execute  $a_t \leftarrow \langle a_{t,1}, \dots, a_{t,N} \rangle$ 
9:     Observe global reward  $r_t, z_{t+1}$ , and  $s_{t+1}$ 
10:    Store  $e_t \leftarrow \langle s_t, a_t, s_{t+1}, r_t, z_{t+1}, p_t \rangle$  in  $E$ 
11:    Refine  $\theta$  and  $\omega$  to minimize  $L_{STEP}$  for all  $e_t \in E$  (Eq. 3)
    $\triangleright$  centralized learning with shared loss  $L_{STEP}$  and buffer  $E$ 

```

In the (centralized) learning step, a function approximator² $\hat{\pi}_\theta = \langle \hat{\pi}_{1,\theta_1}, \dots, \hat{\pi}_{N,\theta_N} \rangle$ with parameters $\theta = \langle \theta_1, \dots, \theta_N \rangle$ is used to approximate π^* (Fig. 1a, orange dashed rectangle). \hat{V}_ω with parameter vector ω is used to approximate the global value function V^* (Fig. 1a, blue component). All approximators minimize the *shared loss* L_{STEP} for all $e_t \in E$ w.r.t. θ and ω ³:

$$L_{STEP} = (y_t - \hat{V}_\omega(s_t))^2 - \frac{1}{N} \sum_{i \in \mathcal{D}} p(a_{t,i}|s_t)^\top \log(\hat{\pi}_{i,\theta_i}(a_{t,i}|\tau_{t,i})) \quad (3)$$

where $y_t = r_t + \gamma \hat{V}_\omega(s_{t+1})$ is the TD target for \hat{V}_ω [43]. $p(a_{t,i}|s_t)$ and $\hat{\pi}_{i,\theta_i}(a_{t,i}|\tau_{t,i})$ are $|\mathcal{A}_i|$ -dimensional probability vectors. The first term is the squared TD error and the second term is the cross entropy between $p(a_{t,i}|s_t)$ and $\hat{\pi}_{i,\theta_i}(a_{t,i}|\tau_{t,i})$.

The formulation of STEP is given in Algorithm 2, where Ψ is a decentralized planning algorithm (e.g., DOLUCT), \hat{M} is the generative model, N is the number of agents, n_b is the computation budget, and $\hat{\pi}_\theta$ and \hat{V}_ω are the function approximators for π^* and V^* respectively. The full architecture of STEP is shown in Fig. 1.

We recommend *centralized* learning to accelerate training and to exploit global knowledge like the state s_t , the joint policy $\hat{\pi}_\theta$, and the action frequencies p_t of all agents to produce coordinated policies, since training usually takes place in a controlled or simulated environment [11, 24, 45]. The amount of data stored in E scales linearly with N . Note that unlike in [12, 24, 36, 41], the function representations of \hat{V}_ω and $\hat{\pi}_{i,\theta_i}$ as illustrated in Fig. 1b are independent of N and can remain unchanged when varying N . While learning is *centralized*, the planning step is *decentralized* to improve the joint policy without searching the joint action space. The planning and learning step only need to be synchronized during *training*. When the MAS is deployed into *production*, the learned policies $\hat{\pi}_{i,\theta_i}$ can be used in a decentralized way without requiring a central instance, global information, or an additional planner anymore.

4.4 Policy and Value Bias Regulation

Estimating π^* and V^* with $\hat{\pi}_\theta$ and \hat{V}_ω respectively induces a bias, thus includes approximation errors in the planning step.

The action selection bias of $\hat{\pi}_{i,\theta_i}$ can be reduced by increasing n_b . The more node Nd_t is visited, the smaller the exploration term multiplied with c in Eq. 2 becomes, which decreases the influence of $\hat{\pi}_{i,\theta_i}$. This causes the search to focus on nodes with higher expected return $\mathbb{E}[G_t|Nd_t]$, which expands the search tree into these directions. The increasing tree depth k discounts the value estimate $\hat{V}_\omega(s_{t+k})$ of newly added nodes Nd_{t+k} by a factor of γ^k , which reduces the bias of \hat{V}_ω for frequently visited paths, if $\gamma < 1$.

5 EXPERIMENTAL SETUP

5.1 Evaluation Environments

Pursuit & Evasion (PE) is a well-known benchmark problem for MARL [15, 45, 46]. We implemented this domain as 8×8 grid (and as 100×100 grid which we denote by PE*, see Table 1) with N *pursuers* as learning agents, N *evaders* as randomly moving entities, and some obstacles as shown in Fig. 3a (note: PE* has *no* obstacles),

²Any machine learning model (e.g., neural network, random forest, ...) could be used as function approximator for $\hat{\pi}$ and \hat{V} , since STEP is not necessarily gradient-based.

³We recommend to use local policy approximations instead of a joint policy. We also experimented with separate losses but the policies tended to overfit and performed poorly in the evaluations, indicating that the shared loss has a regularizing effect [39].

Table 1: Problem-size features of the domains PE and SF.

	# states $ \mathcal{S} $	# joint actions $ \mathcal{A} $
PE (# agents $N = 2$)	$\approx 4.1 \cdot 10^6$	$5^2 = 25$
PE (# agents $N = 4$)	$\approx 1.7 \cdot 10^{13}$	$5^4 = 625$
SF (# agents $N = 4$)	$\approx 6.9 \cdot 10^{53}$	$6^4 = 1,296$
SF (# agents $N = 8$)	$\approx 4.8 \cdot 10^{107}$	$6^8 \approx 1.7 \cdot 10^6$
SF (# agents $N = 12$)	$\approx 3.3 \cdot 10^{161}$	$6^{12} \approx 2.2 \cdot 10^9$
SF (# agents $N = 16$)	$\approx 2.3 \cdot 10^{215}$	$6^{16} \approx 2.8 \cdot 10^{12}$
PE* (# agents $N = 100$)	10^{800}	$5^{100} \approx 7.89 \cdot 10^{69}$
PE* (# agents $N = 200$)	10^{1600}	$5^{200} \approx 6.22 \cdot 10^{139}$

where the pursuers must collaborate to capture all evaders. All pursuers and evaders have random initial positions and are able to move north, south, west, east, or do nothing. The goal is to capture all evaders as a *joint task*. To capture an evader, two pursuers need to occupy the same cell as the evader which yields a global reward of +1. The pursuers have a view range of 5×5 (Fig. 3a).

The *Smart Factory (SF)* domain consists of a 5×5 grid of machines as shown in Fig. 3b and N agents with each having one *item*, a list of four random tasks $tasks_i$ organized in two *buckets*, and a random initial position. An example from [32] is shown in Fig. 3c. All agents are able to enqueue at their current machine, move north, south, west, east, or do nothing. At every time step, each machine processes one agent in its queue with a reward of -0.25 but does nothing with a probability of 0.1. If a task in its current bucket matches with the machine type, the task is removed from the agent’s task list with a reward of +1. An agent i is *complete*, if $tasks_i = \emptyset$ yielding another reward of +1. For each incomplete agent, a reward of -0.1 is given at every time step. All agents have to *avoid conflicts* to ensure fast completion of all tasks. All agents only know their own tasks and perceive other agents at the same or at adjacent machines (Fig. 3c).

5.2 Learning Algorithms

To evaluate the importance of the key elements of STEP, we implemented different variations of STEP using a default computation budget of $n_b = 128$. A random policy is used as default search prior and default prediction of other agent’s behavior. We always set $\Psi = DOLUCT$ as default decentralized planner (Algorithm 1 & 2).

- *Emergent Value function Approximation for Distributed Environments (EVADE)* [32]: Uses only \hat{V}_ω to enhance Ψ .
- *STEP (with prior)*: Extends *EVADE* by additionally learning $\hat{\pi}_{i,\theta_i}$ as search prior for Ψ (Fig. 1c).
- *STEP (with policy)*: Extends *EVADE* by additionally learning $\hat{\pi}_\theta$ as prediction of other agent’s behavior for Ψ (Fig. 1c).
- *STEP*: Extends *EVADE* by additionally learning $\hat{\pi}_\theta$ as search prior *and* as prediction of other agent’s behavior for Ψ .

Due to the vast state and joint action spaces especially in SF (Table 1), we did not experimentally compare with centralized planning, since the computation and specification of explicit transition probabilities for each state would be infeasible. However, we provide experimental results of STEP applied to decentralized closed-loop MCTS (*STEP (DUCT)*) with $\Psi = DUCT$, see Fig. 2) using $\hat{\pi}_\theta$ as search prior and as prediction of other agent’s behavior.

We also implemented independent Q-Learning (*DQN*), *QMIX*, and *COMA* as described in [12, 36] to compare with STEP. *QMIX* is

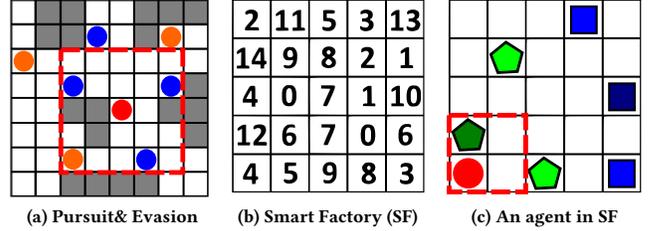


Figure 3: (a) Pursuit & Evasion (PE) with pursuers (red/orange circles) and evaders (blue circles). (b) Machine grid of the Smart Factory (SF) with the numbers in each cell denoting the machine type. (c) An agent i (red circle) with $tasks_i = [\{9, 12\}, \{3, 10\}]$ in the SF of Fig. 3b. It should get processed at the green pentagonal machines first before going to the blue rectangular machines. The red dashed rectangles indicate the red agent’s view range.

an off-policy algorithm which learns a factorization of the joint action value function, while *COMA* is an on-policy algorithm, which uses policy gradient learning with a counterfactual baseline. *DQN*, *QMIX*, and *COMA* are model-free, thus they do not perform any additional search but *QMIX* and *COMA* also exploit global information during training. *DQN* is only trained with local information and has no explicit mechanism for multi-agent credit assignment.

5.3 Function Approximation

We used deep convolutional neural networks⁴ to implement $\hat{\pi}_{i,\theta_i}$ for each agent i and \hat{V}_ω . An experience buffer E was implemented to store the last 10,000 transitions and to sample minibatches of size 32 to perform stochastic gradient descent (SGD) using ADAM with a learning rate of 0.001 to minimize the shared loss L_{STEP} (Eq. 3). We set $\gamma = 0.95$. We also used a *target network* with weights ω^- to generate TD targets [26]. Every 5,000 SGD steps, we set $\omega^- = \omega$.

Since both evaluation domains are gridworlds, the states and local observations can be encoded as multi-channel image as proposed in [15, 32]. The input to the local policy networks $\hat{\pi}_{i,\theta_i}$ is a stacked sequence of local observations of length 5 $\tau_{t,i} = [z_{t-4,i}, \dots, z_{t,i}]$ ⁵ for both domains, while the input to the value network \hat{V}_ω is the global state s_t . The inputs are convolved with 64 filters of size 5×5 with stride 1 followed by three convolutional layers with 64 filters of size 3×3 with stride 1. The output of the final convolutional layer is processed by a fully connected layer with 256 units. The output layer consists of a softmax output for $\hat{\pi}_{i,\theta_i}(a_{t,i}|\tau_{t,i})$ or a single linear output for $\hat{V}_\omega(s_t)$. All hidden layers use ReLU activation.

The local policy components of *DQN*, *QMIX*, and *COMA* had a similar architecture to $\hat{\pi}_{i,\theta_i}$. The central component of *QMIX* and *COMA* had a similar architecture to \hat{V}_ω with the joint action as additional input. With that we tried to ensure a fair comparison w.r.t. the number of learnable parameters.

⁴We also experimented with recurrent neural networks, which yield similar results. However the training time was prohibitively long for a sufficient evaluation.

⁵Integrating actions had no significant effect, thus we did not include them explicitly.

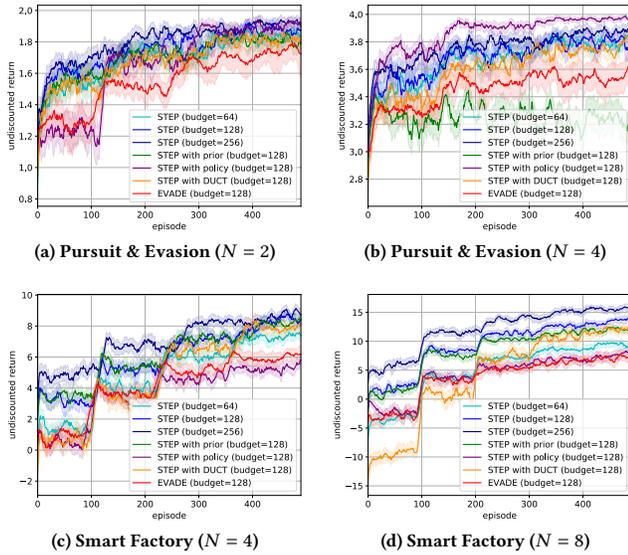


Figure 4: Average training progress of 30 runs of different STEP variations. All planners act in a *fully observable* environment. Shaded areas show the 95 % confidence interval.

6 RESULTS

We conducted various experiments across multiple episodes. An *episode* is reset after 50 time steps, when all evaders are captured (PE), or when all items are complete (SF). We always set $\gamma = 0.95$ and $c = 1$ for decentralized planning. The performance is measured with the undiscounted return in both domains. The problem-size features of all experimental settings are listed in Table 1.

6.1 Ablation Results

We conducted ablation experiments with $N = 2, 4$ agents for PE and $N = 4, 8$ agents for SF to address the following questions regarding the key elements of *STEP*:

- How do the action selection prior $\hat{\pi}_{i,\theta_i}$ and the joint policy $\hat{\pi}_\theta$ (and their absence) affect overall performance?
- Is *STEP* generally able to improve the performance of decentralized planning?
- How well does open-loop planning (*DOLUCT*) perform compared to closed-loop planning (*DUCT*) in these domains?
- How does the budget n_b affect overall performance?

A *run* consists of 500 episodes and is repeated 30 times. All decentralized planners act in a *fully observable* environment (Fig. 1a, green components). The performance results are shown in Fig. 4.

In PE, the joint policy $\hat{\pi}_\theta$ has a stronger influence on the performance than the action selection prior $\hat{\pi}_{i,\theta_i}$, since *STEP (with policy)* outperforms *STEP (with prior)* in the long run. In the 4-agent setting, *STEP (with prior)* fails to converge to a meaningful policy. In SF, *STEP (with prior)* always outperforms *STEP (with policy)*. *STEP* shows competitive performance to *STEP (with policy)* in PE and *STEP (with prior)* in SF and always outperforms *EVADE* and

its closed-loop counterpart *STEP (DUCT)*. Increasing the budget n_b consistently improves the performance of *STEP*.

6.2 State-of-the-Art Comparison

We conducted experiments with $N = 4, 8, 12, 16$ agents for SF and $N = 100, 200$ agents⁶ for PE* (Table 1) to compare the learned local policies of *STEP* (trained with *DOLUCT*) with *DQN*, *QMIX*, and *COMA* to address the following questions:

- How well do the policies learned with *STEP* perform compared to the state-of-the-art and w.r.t. the number of agents?
- How sample efficient is *STEP* w.r.t. training steps?
- How does centralized learning affect overall performance?

In these experiments, only the *trained neural networks* were tested and compared in the *partially observable* environment *without* the support of a decentralized planner and *without* any global information (Fig. 1a, red components).

Since we assume a training simulator (which in our case is the generative model \hat{M} itself), *STEP* requires $n_b \cdot N$ simulated time steps per SGD update. Note that in practice, the planning step can be easily parallelized because we use decentralized planning, thus the actual training time should be much smaller than the sequential time regarded here. However, this experiment focuses on the sample efficiency w.r.t. time steps in the training environment.

COMA requires 20 episodes for each SGD update, while *DQN* and *QMIX* require only one simulated time step per SGD update. Every 50th update, each learned policy was tested on 25 random episodes. A *run* consists of maximum 500,000 simulated time steps (10 million time steps in PE*) to assess the learning stability of *STEP* and is repeated 30 times. For the 4- and 8-agent case, we also provide versions of *STEP* with $n_b \in \{64, 256\}$ from Section 6.1. The performance results of the learned policies are shown in Fig. 5.

QMIX performs best in the 4-agent SF setting and is competitive to *STEP* in the 8-agent setting but is clearly outperformed by *STEP* in settings with $N > 8$ agents. *STEP (budget=64)* is the best *STEP* variant in the 4-agent SF setting with *STEP (budget=128)* slowly keeping up, but in the 8-agent setting, *STEP (budget=128)* becomes competitive to *STEP (budget=64)*. *STEP (budget=256)* is the worst performing *STEP* variant. *COMA* always converges to local optima which are worse than *STEP* and *QMIX*. *DQN* fails to make any stable progress and performs worst in all settings. Compared to model-free algorithms, *STEP* is especially superior when N is large.

7 DISCUSSION

In this paper, we presented *STEP*, a distributed policy iteration scheme to stably learn decentralized policies for cooperative MAS. *STEP* approximates local policies from action recommendations of a decentralized planning algorithm. We proposed a novel centralized training architecture which additionally exploits the simulated training environment to enable Monte Carlo planning on the fully observable problem formulation in order to recommend high quality actions for local policy approximation. For scalable policy improvement, we proposed a decentralized planning scheme which can

⁶While *STEP* and *DQN* can be trained on ordinary machines, *QMIX* and *COMA* require an enormous amount of memory (which probably has to be provided by a super- or cloud computing service) due to the large tensors resulting from the high-dimensional joint action space which are needed to approximate the joint action value function.

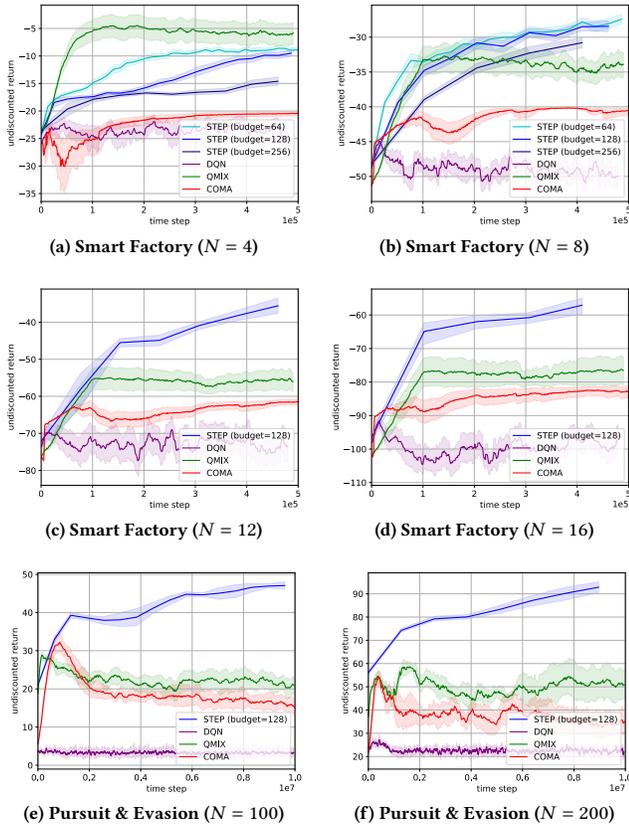


Figure 5: Average test progress of 30 runs of the policies learned with STEP compared to policies learned with model-free MARL w.r.t. the number of training time steps. All learned policies act in a *partially observable* environment. Shaded areas show the 95 % confidence interval.

reintegrate the learned local policies as action selection prior and as prediction of other agents’ behavior for better coordination to further improve the quality of local action recommendations.

The key elements of *STEP* were evaluated in two domains. In PE, the agents have to capture evaders as a *joint task*, while in SF, the agents need to *avoid conflicts* at a shared set of machines. Our experiments show that using the joint policy as a prediction of other agent’s behavior is crucial to coordinate on joint tasks, while using the local policy as a prior is useful to avoid conflicts, when working with shared resources. Combining both elements, results in a more general approach which is able to solve both problems which are common in cooperative MAS. As indicated in Fig. 4b, there are situations, where either the prior or the joint policy dominates all other variations. Finding adequate weighting strategies for each key element would be an interesting direction for future work.

Results from Fig. 4 show that both *DOLUCT* and *DUCT* are able to improve with *STEP*, offering stable policy improvement. *STEP* is able to stably learn decentralized policies, which can be reintegrated into the planning process to further improve and coordinate decentralized planning in contrast to planning with just a random

policy like in the case of *EVADE*. *DUCT* usually requires more computation budget than *DOLUCT* to sufficiently search the closed-loop search space in order to recommend higher quality actions (Fig. 2 and Table 1), thus improving slower than *DOLUCT*.

Increasing the computation budget n_b improves the quality of *STEP* (Fig. 4), which might be due to the decreasing bias when planning with $\hat{\pi}_i, \theta_i$ and \hat{V}_ω (Section 4.4). This could also explain why *DUCT* improves slower than *DOLUCT*: Since *DUCT* needs to explore a much larger search space, it depends more on the bias of $\hat{\pi}_i, \theta_i$ and \hat{V}_ω than *DOLUCT*, especially when n_b is small.

However, increasing n_b requires more simulation steps which can be expensive when scaling up the number of agents (Fig. 5).

Results from Fig. 5 show that *STEP* is able to outperform state-of-the-art approaches to MARL when scaling up w.r.t. the number of agents - even with a small computation budget. While *COMA* and *QMIX* seemingly get stuck in poor local optima due to the high complexity (Table 1) and the difficult credit assignment, *STEP* policies stably improve, showing that decentralized planning is able to further improve the performance of gradient-based MARL. Since model-free approaches work with noisy experience due to stochastic dynamics and the exploration policy, it is harder to learn meaningful policies from the obtained data alone. In contrast, *STEP* is able to provide "clean" training data, by filtering out noisy experience during the planning step. Thus, *STEP* requires less SGD updates than model-free MARL. All centralized learning approaches outperformed *DQN*, which was unable to learn any meaningful policy in all settings, confirming the need for coordination mechanisms during learning rather than naively using independent learning.

Although in Fig. 5a and 5b, *STEP* seems to perform worse with large computation budgets, this is only due to the limited training budget of 500,000 time steps. If the training budget was increased, a larger computation budget n_b would result in even stronger policies as indicated in Fig. 4, 5e, and 5f. Thus, the choice of the computation budget for *STEP* depends on the available amount of training time in practice. Furthermore, the actual training time can be reduced with distributed and parallel computing, since the decentralized planning step of *STEP* is easily parallelizable.

A major limitation of *STEP* is the assumption that all agents behave according their learned local policies during the planning step. While the ablation results in Fig. 4 show a benefit of this assumption compared to purely random simulations in joint task problems, *STEP* could be further improved by integrating opponent modeling or theory of mind mechanisms [11].

Using online planning in production in combination with the approximated policies could further improve performance [38, 39]. To adjust *DOLUCT* for partial observability, the joint belief state needs to be approximated with the joint action, which can be predicted with the learned joint policy (after training all local policies can be replicated on each planner) to perform a decentralized Monte Carlo update of the joint belief state [40]. Since this paper focused on stably learning local policies compared to state-of-the-art MARL, we defer an additional evaluation of enhancing partially observable decentralized planning in partially observable environments with *STEP* to future work.

REFERENCES

- [1] Thomas Anthony, Zheng Tian, and David Barber. 2017. Thinking Fast and Slow with Deep Learning and Tree Search. In *Advances in Neural Information Processing Systems*. 5360–5370.
- [2] Peter Auer, Nicola Cesa-Bianchi, and Paul Fischer. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine learning* 47, 2-3 (2002), 235–256.
- [3] Daniel S Bernstein, Christopher Amato, Eric A Hansen, and Shlomo Zilberstein. 2009. Policy Iteration for Decentralized Control of Markov Decision Processes. *Journal of Artificial Intelligence Research* 34 (2009), 89–132.
- [4] Graeme Best, Oliver M Cliff, Timothy Patten, Ramgopal R Mettu, and Robert Fitch. 2019. Dec-MCTS: Decentralized Planning for Multi-Robot Active Perception. *The International Journal of Robotics Research* 38, 2-3 (2019), 316–337.
- [5] Craig Boutilier. 1996. Planning, Learning and Coordination in Multiagent Decision Processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*. Morgan Kaufmann Publishers Inc., 195–210.
- [6] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. 2010. Multi-Agent Reinforcement Learning: An Overview. In *Innovations in multi-agent systems and applications-1*. Springer, 183–221.
- [7] Yu-Han Chang, Tracey Ho, and Leslie P Kaelbling. 2004. All Learning is Local: Multi-Agent Learning in Global Reward Games. In *Advances in neural information processing systems*. 807–814.
- [8] Daniel Claes, Frans Oliehoek, Hendrik Baier, and Karl Tuyls. 2017. Decentralised Online Planning for Multi-Robot Warehouse Commissioning. In *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS.
- [9] Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. 2004. Approximate Solutions for Partially Observable Stochastic Games with Common Payoffs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004*. IEEE, 136–143.
- [10] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. 2016. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems*. 2137–2145.
- [11] Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. 2018. Learning with Opponent-Learning Awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*.
- [12] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2018. Counterfactual Multi-Agent Policy Gradients. In *AAAI*.
- [13] Carlos Guestrin, Daphne Koller, and Ronald Parr. 2002. Multiagent Planning with Factored MDPs. In *Advances in neural information processing systems*. 1523–1530.
- [14] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. 2014. Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning. In *Advances in Neural Information Processing Systems*.
- [15] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. 2017. Cooperative Multi-Agent Control using Deep Reinforcement Learning. In *International Conference on Autonomous Agents and Multiagent Systems*. Springer, 66–83.
- [16] Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. 2004. Dynamic Programming for Partially Observable Stochastic Games. In *Proceedings of the 19th National Conference on Artificial Intelligence*.
- [17] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. 2019. A Survey and Critique of Multiagent Deep Reinforcement Learning. *Autonomous Agents and Multi-Agent Systems* (2019), 1–48.
- [18] Ronald A. Howard. 1961. *Dynamic Programming and Markov Processes*.
- [19] Bilal Kartal, Pablo Hernandez-Leal, and Matthew E Taylor. 2019. Action Guidance with MCTS for Deep Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.
- [20] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based Monte-Carlo Planning. In *European conference on machine learning*. Springer, 282–293.
- [21] Jelle R Kok and Nikos Vlassis. 2004. Sparse Cooperative Q-learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*. 61.
- [22] Erwan Lecarpentier, Guillaume Infantes, Charles Lesire, and Emmanuel Rachelson. 2018. Open Loop Execution of Tree-Search Algorithms. In *Proceedings of the 27th IJCAI*. AAAI Press, 2362–2368.
- [23] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. 2017. Multi-Agent Reinforcement Learning in Sequential Social Dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, 464–473.
- [24] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems*. 6379–6390.
- [25] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. [n. d.]. Hysteretic Q-Learning: An Algorithm for Decentralized Reinforcement Learning in Cooperative Multi-Agent Teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 64–69.
- [26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, et al. 2015. Human-Level Control through Deep Reinforcement Learning. *Nature* (2015).
- [27] Frans A Oliehoek and Christopher Amato. 2016. *A Concise Introduction to Decentralized POMDPs*. Vol. 1. Springer.
- [28] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. 2017. Deep Decentralized Multi-task Multi-Agent Reinforcement Learning under Partial Observability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2681–2690.
- [29] Gregory Palmer, Karl Tuyls, Daan Bloembergen, and Rahul Savani. 2018. Lenient Multi-Agent Deep Reinforcement Learning. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS.
- [30] Liviu Panait, Keith Sullivan, and Sean Luke. 2006. Lenient Learners in Cooperative Multiagent Systems. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM, 801–803.
- [31] Diego Perez Liebana, Jens Dieskau, Martin Hunermund, Sanaz Mostaghim, and Simon Lucas. 2015. Open Loop Search for General Video Game Playing. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 337–344.
- [32] Thomy Phan, Lenz Belzner, Thomas Gabor, and Kyrill Schmid. 2018. Leveraging Statistical Multi-Agent Online Planning with Emergent Value Function Approximation. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, 730–738.
- [33] Thomy Phan, Lenz Belzner, Marie Kiermeier, Markus Friedrich, Kyrill Schmid, and Claudia Linnhoff-Popien. 2019. Memory Bounded Open-Loop Planning in Large POMDPs Using Thompson Sampling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7941–7948.
- [34] Thomy Phan, Thomas Gabor, Robert Müller, Christoph Roch, and Claudia Linnhoff-Popien. 2019. Adaptive Thompson Sampling Stacks for Memory Bounded Open-Loop Planning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 5607–5613.
- [35] Thomy Phan, Kyrill Schmid, Lenz Belzner, Thomas Gabor, Sebastian Feld, and Claudia Linnhoff-Popien. 2019. Distributed Policy Iteration for Scalable Approximation of Cooperative Multi-Agent Policies. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (Extended Abstract)*. IFAAMAS, 2162–2164.
- [36] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In *International Conference on Machine Learning*. 4292–4301.
- [37] Sven Seuken and Shlomo Zilberstein. 2007. Memory-Bounded Dynamic Programming for DEC-POMDPs. In *IJCAI*. 2009–2015.
- [38] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharmsharan Kumaran, Thore Graepel, et al. 2018. A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through Self-Play. *Science* 362, 6419 (2018), 1140–1144.
- [39] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the Game of Go without Human Knowledge. *Nature* 550, 7676 (2017), 354.
- [40] David Silver and Joel Veness. 2010. Monte-Carlo Planning in Large POMDPs. In *Advances in neural information processing systems*. 2164–2172.
- [41] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. 2019. QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning. In *International Conference on Machine Learning*. 5887–5896.
- [42] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, et al. 2018. 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 (Extended Abstract)*. IFAAMAS, 2085–2087.
- [43] Richard S Sutton and Andrew G Barto. 1998. *Introduction to Reinforcement Learning*.
- [44] Ardi Tampuu, Tanel Matiisen, Dorian Kodolja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. 2017. Multiagent Cooperation and Competition with Deep Reinforcement Learning. *PLoS one* 12, 4 (2017), e0172395.
- [45] Ming Tan. 1993. Multi-Agent Reinforcement Learning: Independent versus Cooperative Agents. In *Proc. of the 10th International Conference on International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 330–337.
- [46] Rene Vidal, Omid Shakernia, H Jin Kim, David Hyunchul Shim, and Shankar Sastry. 2002. Probabilistic Pursuit-Evasion Games: Theory, Implementation, and Experimental Evaluation. *IEEE transactions on robotics and automation* 18, 5 (2002), 662–669.
- [47] Ari Weinstein and Michael L Littman. 2013. Open-Loop Planning in Large-Scale Stochastic Domains. In *Twenty-Seventh AAAI*.
- [48] David H Wolpert and Kagan Tumer. 2002. Optimal Payoff Functions for Members of Collectives. In *Modeling complexity in economic and social systems*. World Scientific, 355–369.
- [49] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. 2009. Multi-agent Online Planning with Communication. In *Proceedings of the 19th ICAPS*. AAAI Press, 321–328.