MAVIPER: Learning Decision Tree Policies for Interpretable Multi-Agent Reinforcement Learning

Stephanie Milani*[1]✉, Zhicheng Zhang*[2],
Nicholay Topin[1], Zheyuan Ryan Shi[1], Charles Kamhoua[3],
Evangelos E. Papalexakis[4], and Fei Fang[1]

[1] Carnegie Mellon University smilani@andrew.cmu.edu
[2] Shanghai Jiao Tong University
[3] Army Research Lab
[4] University of California, Riverside

Abstract. Many recent breakthroughs in multi-agent reinforcement learning (MARL) require the use of deep neural networks, which are challenging for human experts to interpret and understand. On the other hand, existing work on interpretable reinforcement learning (RL) has shown promise in extracting more interpretable decision tree-based policies from neural networks, but only in the single-agent setting. To fill this gap, we propose the first set of algorithms that extract interpretable decision-tree policies from neural networks trained with MARL. The first algorithm, IVIPER, extends VIPER, a recent method for single-agent interpretable RL, to the multi-agent setting. We demonstrate that IVIPER learns high-quality decision-tree policies for each agent. To better capture coordination between agents, we propose a novel centralized decision-tree training algorithm, MAVIPER. MAVIPER jointly grows the trees of each agent by predicting the behavior of the other agents using their anticipated trees, and uses resampling to focus on states that are critical for its interactions with other agents. We show that both algorithms generally outperform the baselines and that MAVIPER-trained agents achieve better-coordinated performance than IVIPER-trained agents on three different multi-agent particle-world environments.

Keywords: interpretability · explainability · multi-agent reinforcement learning

1 Introduction

Multi-agent reinforcement learning (MARL) is a promising technique for solving challenging problems, such as air traffic control [5], train scheduling [27], cyber defense [22], and autonomous driving [4]. In many of these scenarios, we want to train a *team* of cooperating agents. Other settings, like cyber defense, involve an

* Equal contribution

adversary or set of adversaries with goals that may be at odds with the team of defenders. To obtain high-performing agents, most of the recent breakthroughs in MARL rely on neural networks (NNs) [10, 35], which have thousands to millions of parameters and are challenging for a person to interpret and verify. Real-world risks necessitate learning *interpretable* policies that people can inspect and verify before deployment, while still performing well at the specified task and being robust to a variety of attackers (if applicable).

Decision trees [34] (DTs) are generally considered to be an *intrinsically* interpretable model family [28]: sufficiently small trees can be contemplated by a person at once (simulatability), have subparts that can be intuitively explained (decomposability), and are verifiable (algorithmic transparency) [18]. In the RL setting, DT-like models have been successfully used to model transition functions [40], reward functions [8], value functions [33, 43], and policies [24]. Although learning DT policies for interpretability has been investigated in the single-agent RL setting [24, 32, 37], it has yet to be explored in the multi-agent setting.

To address this gap, we propose two algorithms, IVIPER and MAVIPER, which combine ideas from model compression and imitation learning to learn DT policies in the multi-agent setting. Both algorithms extend VIPER [2], which extracts DT policies for single-agent RL. IVIPER and MAVIPER work with most existing NN-based MARL algorithms: the policies generated by these algorithms serve as "expert policies" and guide the training of a set of DT policies.

The main contributions of this work are as follows. First, we introduce the IVIPER algorithm as a novel extension of the single-agent VIPER algorithm to multi-agent settings. Indeed, IVIPER trains DT policies that achieve high individual performance in the multi-agent setting. Second, to better capture coordination between agents, we propose a novel centralized DT training algorithm, MAVIPER. MAVIPER jointly grows the trees of each agent by predicting the behavior of the other agents using their anticipated trees. To train each agent's policy, MAVIPER uses a novel resampling scheme to find states that are considered critical for its interactions with other agents. We show that MAVIPER-trained agents achieve better coordinated performance than IVIPER-trained agents on three different multi-agent particle-world environments.

2 Background and Preliminaries

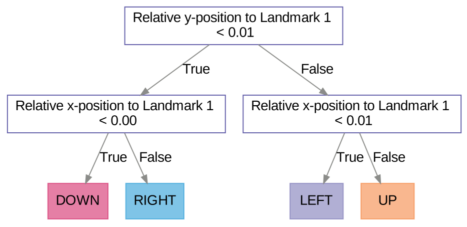
We focus on the problem of learning interpretable DT policies in the multi-agent setting. We first describe the formalism of our multi-agent setting, then discuss DT policies and review the single-agent version of VIPER.

2.1 Markov Games and MARL Algorithms

In MARL, agents act in an environment defined by a Markov game [19, 38]. A Markov game for N agents consists of a set of states $\mathcal{S}$ describing all possible configurations for all agents, the initial state distribution $\rho : \mathcal{S} \to [0, 1]$, and the set of actions $\mathcal{A}_1, ..., \mathcal{A}_N$ and observations $\mathcal{O}_1, ..., \mathcal{O}_N$ for each agent $i \in [N]$. Each agent aims to maximize its own total expected return $R_i = \sum_{t=0}^{\infty} \gamma^t r_i^t$, where γ is the discount factor that weights the relative importance of future rewards. To do so, each agent selects actions using a policy $\pi_{\theta_i} : \mathcal{O}_i \to \mathcal{A}_i$. After the agents simultaneously execute their actions $\overrightarrow{a}$ in the environment, the environment produces the next state according to the state transition function $P : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \to \mathcal{S}$. Each agent i receives reward according to a reward function $r_i : \mathcal{S} \times \mathcal{A}_i \to \mathbb{R}$ and a private observation, consisting of a vector of *features*, correlated with the state $o_i : \mathcal{S} \to \mathcal{O}_i$.

Given a policy profile $\pi = (\pi_1, ..., \pi_N)$, agent i's value function is defined as: $V_i^\pi(s) = r_i + \gamma \sum_{s' \in \mathcal{S}} P(s, \pi_1(o_1), ..., \pi_N(o_N), s') V_i^\pi(s')$ and state-action value function is: $Q_i^\pi(s, a_1, ..., a_N) = r_i + \gamma \sum_{s' \in S} P(s, a_1, ..., a_N, s') V_i^\pi(s')$. We refer to a policy profile excluding agent i as π_{-i}.

MARL algorithms fall into two categories: value-based [35, 39, 41] and actor-critic [11, 16, 20, 48]. Value-based methods often approximate Q-functions for individual agents in the form of $Q_i^\pi(o_i, a_i)$ and derive the policies π_i by taking actions with the maximum Q-values. In contrast, actor-critic methods often follow the centralized training and decentralized execution (CTDE) paradigm [30]. They train agents in a centralized manner, enabling agents to leverage information beyond their private observation during training; however, agents must behave in a decentralized manner during execution. Each agent i uses a centralized critic network Q_i^π, which takes as input some state information x (including the observations of all agents) and the actions of all agents. This assumption addresses the stationarity issue in MARL training: without access to the actions of other agents, the environment appears non-stationary from the perspective of any one agent. Each agent i also has a policy network π_i that takes as input its observation o_i.


Relative y-position to Landmark 1
< 0.01
True
False
Relative x-position to Landmark 1
< 0.00
Relative x-position to Landmark 1
< 0.01
True
False
True
False
DOWN
RIGHT
LEFT
UP
Fig. 1: A decision tree of depth two that MAVIPER learns in the Cooperative Navigation environment. The learned decision tree captures the expert's behavior of going to one of the landmarks.

2.2 Decision Tree Policies

DTs are tree-like models that recursively partition the input space along a specific feature using a cutoff value. These models produce axis-parallel partitions: internal nodes are the intermediate partitions, and leaf nodes are the final partitions. When used to represent policies, the internal nodes represent the features and values of the input state that the agent uses to choose its action, and the leaf nodes correspond to chosen actions given some input state. For an example of a DT policy, see Figure 1.

2.3 VIPER

VIPER [2] is a popular algorithm [7, 21, 25] that extracts DT policies for a finite-horizon Markov decision process given an *expert* policy trained using any single-agent RL algorithm. It combines ideas from model compression [6, 13] and imitation learning [1] — specifically, a variation of the DAGGER algorithm [36]. It uses a high-performing deep NN that approximates the state-action value function to guide the training of a DT policy.

VIPER trains a DT policy $\hat{\pi}^m$ in each iteration m; the final output is the best policy among all iterations. More concretely, in iteration m, it samples K trajectories $\{(s, \hat{\pi}^{m-1}(s)) \sim d^{\hat{\pi}^{m-1}}\}$ following the DT policy trained at the previous iteration. Then, it uses the expert policy π^* to suggest actions for each visited state, leading to the dataset $D^m = \{(s, \pi^*(s)) \sim d^{\hat{\pi}^{m-1}}\}$ (Line 4, Alg. 3). VIPER adds these relabeled experiences to a dataset $\mathcal{D}$ consisting of experiences from previous iterations. Let V^{π^*} and Q^{π^*} be the state value function and state-action value function given the expert policy π^*. VIPER resamples points $(s, a) \in \mathcal{D}$ according to weights: $\tilde{l}(s) = V^{\pi^*}(s) - \min_{a \in A} Q^{\pi^*}(s, a)$. See Algorithm 3 in Appendix A for the full VIPER algorithm.

3 Approach

We present two algorithms: IVIPER and MAVIPER. Both are general policy extraction algorithms for the multi-agent setting inspired by the single-agent VIPER algorithm. At a high level, given an expert policy profile $\pi^* = (\pi_1^*, ...\pi_N^*)$ with associated state-action value functions $Q^{\pi^*} = (Q_1^{\pi^*}, ..., Q_N^{\pi^*})$ trained by an existing MARL algorithm, both algorithms produce a DT policy $\hat{\pi}_i$ for each agent i. These algorithms work with various state-of-art MARL algorithms, including value-based and multi-agent actor-critic methods. We first discuss IVIPER, the basic version of our multi-agent DT learning algorithm. We then introduce additional changes that form the full MAVIPER algorithm.

3.1 IVIPER

Motivated by the practical success of single-agent RL algorithms in the MARL setting [23, 3], we extend single-agent VIPER to the multi-agent setting by independently applying the single-agent algorithm to each agent, with a few critical changes described below. Algorithm 1 shows the full IVIPER pseudocode.

First, we ensure that each agent has sufficient information for training its DT policy. Each agent has its own dataset $\mathcal{D}_i$ of training tuples. When using VIPER with multi-agent actor-critic methods that leverage a per-agent centralized critic network Q_i^π, we ensure that each agent's dataset $\mathcal{D}_i$ has not only its observation and actions, but also the complete state information x — which consists of the observations of all of the agents — and the expert-labeled actions of all of the other agents $\pi_j^*(o_j) \forall j \neq i$. By providing each agent with the information about all other agents, we avoid the stationarity issue that arises when the policies of all agents are changing throughout the training process (like in MARL).

Algorithm 1 IVIPER in Multi-Agent Setting

```
Input: $(X, A, P, R)$, $\pi^*$ , $Q^{\pi^*} = (Q_1^{\pi^*}, ..., Q_N^{\pi^*})$, $K$, $M$
Output: $\hat{\pi}_1, ..., \hat{\pi}_N$
1: for i=1 to N do
2:     Initialize dataset $\mathcal{D}_i \leftarrow \emptyset$ and policy $\hat{\pi}_i^0 \leftarrow \pi_i^*$
3:     for $m = 1$ to $M$ do
4:         Sample $K$ trajectories: $\mathcal{D}_i^m \leftarrow \{(x, \pi_1^*(o_1), ..., \pi_N^*(o_N)) \sim d^{\hat{\pi}_i^{m-1}, \pi_{-i}^*}\}$
5:         Aggregate dataset $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \mathcal{D}_i^m$
6:         Resample dataset according to loss:
             $\mathcal{D}_i' \leftarrow \{(x, \overrightarrow{a}) \sim p((x, \overrightarrow{a})) \propto \tilde{l}_i(x)\mathbb{I}[(x, \overrightarrow{a}) \in \mathcal{D}_i]\}$
7:         Train decision tree $\hat{\pi}_i^m \leftarrow$ TrainDecisionTree($\mathcal{D}_i'$)
8:     Get best policy $\hat{\pi}_i \leftarrow$ BestPolicy($\hat{\pi}_i^1, ..., \hat{\pi}_i^M, \pi_{-i}^*$)
9: return Best policies for each agent $\hat{\pi} = (\hat{\pi}_1, ..., \hat{\pi}_N)$
```

Second, we account for important changes that emerge from moving to a multi-agent formalism. When we sample and relabel trajectories for training each agent's DT policy, we sample from the distribution $d^{\hat{\pi}_i^{m-1}, \pi_{-i}^*}$ induced by agent i's policy at the previous iteration $\hat{\pi}_i^{m-1}$ and the expert policies of all other agents π_{-i}^*. We only relabel the action for agent i because the other agents choose their actions according to π^*. It is equivalent to treating all other expert agents as part of the environment and only using DT policy for agent i.

Third, we incorporate the actions of all agents when resampling the dataset to construct a new, weighted dataset (Line 6, Algorithm 1). If the MARL algorithm uses a centralized critic $Q(s, \overrightarrow{a})$, we resample points according to:

$$p((x, a_1, ..., a_N)) \propto \tilde{l}_i(x)\mathbb{I}[(x, a_1, ..., a_N) \in \mathcal{D}_i], \tag{1}$$

where,

$$\tilde{l}_i(x) = V_i^{\pi^*}(x) - \min_{a_i \in \mathcal{A}_i} Q_i^{\pi^*}(x, a_i, \overrightarrow{a}_{-i})|_{\overrightarrow{a}_{-i} = \pi_j^*(o_j) \forall j \neq i}. \tag{2}$$

Crucially, we include the actions of all other agents in Equation (2) to select agent i's minimum Q-value from its centralized state-action value function.

When applied to value-based methods, IVIPER is more similar to single-agent VIPER. In particular, in Line 4, Algorithm 1, it is sufficient to only store o_i and $\pi_i^*(o_i)$ in the dataset $\mathcal{D}_i^m$, although we still must sample trajectories according to $\hat{\pi}_i^{m-1}$ and π_{-i}^*. In Line 6, we use $\tilde{l}(x) = V_i^{\pi^*}(s) - \min_{a_i \in \mathcal{A}_i} Q_i^{\pi^*}(o_i, a_i)$ from single-agent VIPER, removing the reliance of the loss on a centralized critic.

Taken together, these algorithmic changes form the basis of the IVIPER algorithm. This algorithm can be viewed as transforming the multi-agent learning problem to a single-agent one, in which other agents are folded into the environment. This approach works well if i) we only want an interpretable policy for a single agent in a multi-agent setting or ii) agents do not need to *coordinate* with each other. When coordination is needed, this algorithm does not reliably capture coordinated behaviors, as each DT is trained independently without consideration for what the other agent's resulting DT policy will learn. This issue is particularly apparent when trees are constrained to have a small maximum depth, as is desired for interpretability.

3.2 MAVIPER

To address the issue of coordination, we propose MAVIPER, our novel algorithm for centralized training of coordinated multi-agent DT policies. For expository purpose, we describe MAVIPER in a fully cooperative setting, then explain how to use MAVIPER for mixed cooperative-competitive settings. At a high-level, MAVIPER trains all of the DT policies, one for each agent, in a centralized manner. It jointly grows the trees of each agent by predicting the behavior of the other agents in the environment using their anticipated trees. To train each DT policy, MAVIPER employs a new resampling technique to find states that are critical for its interactions with other agents. Algorithm 2 shows the full MAVIPER algorithm. Specifically, MAVIPER is built upon the following extensions to IVIPER that aim at addressing the issue of coordination.

First, MAVIPER does not calculate the probability $p(x)$ of a joint observation x by viewing the other agents as stationary experts. Instead, MAVIPER focuses on the critical states where a good joint action can make a difference. Specifically, MAVIPER aims to measure how much worse off agent i would be, taking expectation over all possible joint actions of the other agents, if it acts in the worst way possible compared with when it acts in the same way as the expert agent. So, we define $l_i(x)$, as in Equation (2), as:

$$\tilde{l}_i(x) = \mathbb{E}_{\overrightarrow{a}_{-i}} \left[Q_i^{\pi^*}(x, \pi_i^*(x), \overrightarrow{a}_{-i}) - \min_{a_i \in \mathcal{A}_i} Q_i^{\pi^*}(x, a_i, \overrightarrow{a}_{-i}) \right]. \tag{3}$$

MAVIPER uses the DT policies $(\hat{\pi}_1^{m-1}, \ldots, \hat{\pi}_N^{m-1})$ from the last iteration to perform rollouts and collect new data.

Second, we add a prediction module to the DT training process to increase the *joint* accuracy, as shown in the `Predict` function. The goal of the prediction module is to predict the actions that the other DTs $\{\hat{\pi}_j\}_{j \neq i}$ might make, given their partial observations. To make the most of the prediction module, MAVIPER grows the trees evenly using a breadth-first ordering to avoid biasing towards the result of any specific tree. Since the trees are not complete at the time of prediction, we use the output of another DT trained with the full dataset associated with that node for the prediction. Following the intuition that the correct prediction of one agent alone may not yield much benefit if the other agents are wrong, we use this prediction module to remove all data points whose proportion of correct predictions is lower than a predefined threshold. We then calculate the splitting criteria based on this modified dataset and continue iteratively growing the tree.

In some mixed cooperative-competitive settings, agents in a team share goals and need to coordinate with each other, but they face other agents or other teams whose goals are not fully aligned with theirs. In these settings, MAVIPER follows a similar procedure to jointly train policies for agents in the same team

Algorithm 2 MAVIPER (Joint Training)

```
Input: $(\mathcal{X}, A, P, R)$, $\pi^*$, $Q^{\pi^*} = (Q_1^{\pi^*}, \ldots, Q_N^{\pi^*})$, $K$, $M$
Output: $(\hat{\pi}_1, \ldots, \hat{\pi}_N)$
 1: Initialize dataset $\mathcal{D} \leftarrow \emptyset$ and policy for each agent $\hat{\pi}_i^0 \leftarrow \pi_i^* \; \forall i \in N$
 2: for $m = 1$ to $M$ do
 3:     Sample $K$ trajectories: $\mathcal{D}^m \leftarrow \{(x, \pi_1^*(o_1), \ldots, \pi_N^*(o_N)) \sim d^{(\hat{\pi}_1^{m-1}, \ldots, \hat{\pi}_N^{m-1})}\}$
 4:     Aggregate dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}^m$
 5:     For each agent $i$, resample $\mathcal{D}_i$ according to loss:
          $\mathcal{D}_i \leftarrow \{(x, \vec{a}) \sim p((x, \vec{a})) \propto \tilde{l}_i(x)\mathbb{I}[(x, \vec{a}) \in \mathcal{D}]\} \forall i \in N$
 6:     Jointly train DTs: $(\hat{\pi}_1^m, \ldots, \hat{\pi}_N^m) \leftarrow$ TrainJointTrees$(\mathcal{D}_1, \ldots, \mathcal{D}_N)$
 7: return Best set of agents $\hat{\pi} = (\hat{\pi}_1, \ldots, \hat{\pi}_N) \in \{(\hat{\pi}_1^1, \ldots, \hat{\pi}_N^1), \ldots, (\hat{\pi}_1^M, \ldots, \hat{\pi}_N^M)\}$

 8: function TRAINJOINTTREES$(\mathcal{D}_1, \ldots, \mathcal{D}_N)$
 9:     Initialize decision trees $\hat{\pi}_1^m, \ldots, \hat{\pi}_N^m$.
10:     repeat
11:         Grow one more level for agent $i$'s tree $\hat{\pi}_i^m \leftarrow$ Build$(\hat{\pi}_1^m, \ldots, \hat{\pi}_N^m, \mathcal{D}_i)$
12:         Move to the next agent: $i \leftarrow (i+1)\%N$
13:     until all trees have grown to the maximum depth allowed
14:     return decision trees $\hat{\pi}_1^m, \ldots, \hat{\pi}_N^m$

15: function BUILD$(\hat{\pi}_1^m, \ldots, \hat{\pi}_N^m, \mathcal{D}_i)$
16:     for each data point $(x, \vec{a}) \in \mathcal{D}_i$ do
17:         // Will agent $j$'s (projected) final DT predict its action correctly?
18:         $v_j \leftarrow \mathbb{I}\left[\text{Predict}(\hat{\pi}_j^m, x) = a_j\right] \forall j \in [1, N]$
19:         // This data point is useful only if many agents' final DTs predict correctly.
20:         if $\sum_{j=1}^N v_j <$ threshold then Remove $d$ from dataset: $\mathcal{D}_i \leftarrow \mathcal{D}_i \setminus \{(x, \vec{a})\}$
21:     $\hat{\pi}_i^m \leftarrow$ Calculate best next feature split for DT $\hat{\pi}_i^m$ using $\mathcal{D}_i$.
22:     return $\hat{\pi}_i^m$

23: function PREDICT$(\hat{\pi}_j^m, x)$
24:     Use $x$ to traverse $\hat{\pi}_j^m$ until leaf node $l(x)$
25:     Train a projected final DT $\hat{\pi}_j' \leftarrow$ TrainDecisionTree$(\mathcal{D}_j)$
26:     return $\pi$.predict$(x)$
```

to ensure coordination. More specifically, for a team Z, the `Build` and `Predict` function is constrained to only make predictions for the agents in the same team. Equation (3) now takes the expectation over the joint actions for agents outside the team and becomes:

$$\tilde{l}_i(x) = \mathbb{E}_{\overrightarrow{a}_{-Z}} \left[Q_i^{\pi^*}(x, \pi_i^*(x), \overrightarrow{a}_{-Z}) - \min_{a_i \in \mathcal{A}_i} Q_i^{\pi^*}(x, a_i, \overrightarrow{a}_{-Z}) \right]. \quad (4)$$

Taken together, these changes comprise the MAVIPER algorithm. Because we explicitly account for the anticipated behavior of other agents in both the predictions and the resampling probability, we hypothesize that MAVIPER will better capture coordinated behavior.

4 Experiments

We now investigate how well MAVIPER and IVIPER agents perform in a variety of environments. Because the goal is to learn high-performing yet interpretable policies, we evaluate the quality of the trained policies in three multi-agent environments: two mixed competitive-cooperative environments and one fully cooperative environment. We measure how well the DT policies perform in the environment because our goal is to *deploy* these policies, not the expert ones.

Since small DTs are considered interpretable, we constrain the maximum tree depth to be at most 6. The expert policies used to guide the DT training are generated by MADDPG [20][*]. We compare to two baselines:

1. Fitted Q-Iteration. We iteratively approximate the Q-function with a regression DT [9]. We discretize states to account for continuous state values. More details in Appendix B.2. We derive the policy by taking the the action associated with the highest estimated Q-value for that input state.
2. Imitation DT. Each DT policy is directly trained using a dataset collected by running the expert policies for multiple episodes. No resampling is performed. The observations for an agent are the features, and the actions for that agent are the labels.

We detail the hyperparameters and the hyperparameter-selection process in Appendix B.3. We train a high-performing MADDPG expert, then run each DT-learning algorithm 10 times with different random seeds. We evaluate all policies by running 100 episodes. Error bars correspond to the 95% confidence interval. Our code is available through our project website: `https://stephmilani.github.io/maviper/`.

4.1 Environments

We evaluate our algorithms on three multi-agent particle world environments [20], described below. Episodes terminate when the maximum number of timesteps $T = 25$ is reached. We choose the primary performance metric based on the environment (detailed below), and we also provide results using expected return as the performance metric in Appendix C.

[*] We use the Pytorch [31] implementation `https://github.com/shariqiqbal2810/maddpg-pytorch`.

Physical Deception. In this environment, a team of N defenders must protect N targets from one adversary. One of the targets is the true target, which is known to the defenders but not to the adversary. For our experiments, $N = 2$. Defenders succeed during an episode if they split up to cover all of the targets simultaneously; the adversary succeeds if it reaches the true target during the episode. Covering and reaching targets is defined as being ϵ-close to a target for at least one timestep during the episode. We use the defenders' and the adversary's success rate as the primary performance metric in this environment.

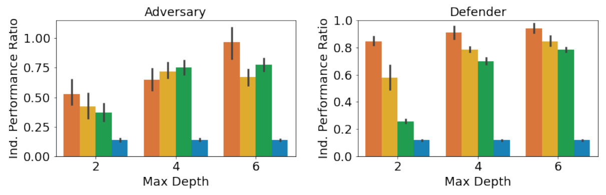
Cooperative Navigation. This environment consists of a team of N agents, who must learn to cover all N targets while avoiding collisions with each other. For our experiments, $N = 3$. Agents succeed during an episode if they split up to cover all of the targets without colliding. Our primary performance metric is the summation of the distance of the closest agent to each target, for all targets. Low values of the metric indicate that the agents correctly learn to split up.

Predator-prey. This variant involves a team of K slower, cooperating predators that chase M faster prey. There are $L = 2$ landmarks impeding the way. We choose $K = M = 2$. We assume that each agent has a restricted observation space mostly consisting of binarized relative positions and velocity (if applicable) of the landmarks and other agents in the environment. See Appendix B.1 for full details. Our primary performance metric is the number of collisions between predators and prey. For prey, lower is better; for predators, higher is better.

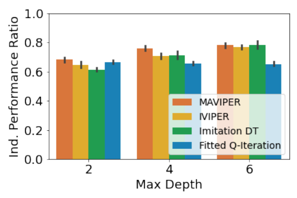
4.2 Results

For each environment, we compare the DT policies generated by different methods and check if IVIPER and MAVIPER agents achieve better performance ratio than the baselines overall. We also investigate whether MAVIPER learns better coordinated behavior than IVIPER. Furthermore, we investigate which algorithms are the most robust to different types of opponents. We conclude with an ablation study to determine which components of the MAVIPER algorithm contribute most to its success.

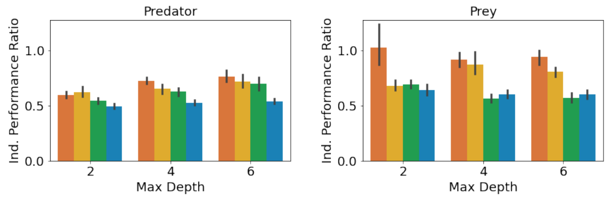
Individual Performance Compared to Experts We analyze the performance of the DT policies when only one agent adopts the DT policy while all other agents use the expert policies. Given a DT policy profile $\hat{\pi}$ and the expert policy profile π^*, if agent i who belongs to team Z uses its DT policy, then the individual performance ratio is defined as: $\frac{U_Z(\hat{\pi}_i, \pi^*_{-i})}{U_Z(\pi^*)}$, where $U_Z(\cdot)$ is team Z's performance given the agents' policy profile (since we define our primary performance metrics at the team level). A performance ratio of 1 means that the DT policies perform as well as the expert ones. We can get a ratio above 1, since we compare the performance of the DT and the expert policies in the environment, not the similarity of the DT and expert policies.



(a) Physical Deception



(b) Cooperative Navigation



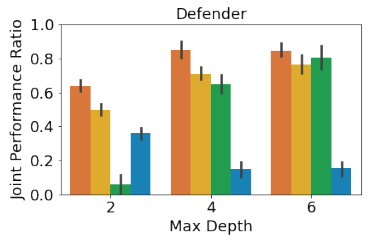
(c) Predator-prey

Fig. 2: Individual performance ratio: Relative performance when only one agent adopts DT policy and all other agents use expert policy.

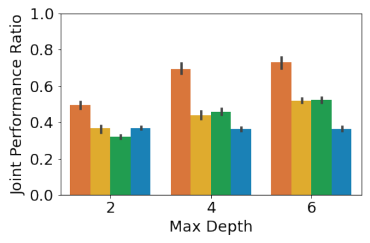
We report the mean individual performance ratio for each team in Figure 2, averaged over all trials and all agents in the team. As shown in Figure 2a, individual MAVIPER and IVIPER defenders outperform the two baselines for all maximum depths in the physical deception environment. However, MAVIPER and IVIPER adversaries perform similarly to the Imitation DT adversary, indicating that the correct strategy may be simple enough to capture with a less-sophisticated algorithm. Agents also perform similarly on the cooperative navigation environment (Figure 2b). As mentioned in the original MADDPG paper [20], this environment has a less stark contrast between success and failure, so these results are not unexpected.

In predator-prey, we see the most notable performance difference when comparing the predator. When the maximum depth is 2, only MAVIPER achieves near-expert performance. When the maximum depths are 4 and 6, MAVIPER and IVIPER agents achieve similar performance and significantly outperform the baselines. The preys achieve similar performance across all algorithms. We suspect that the complexity of this environment makes it challenging to replace even a single prey's policy with a DT.

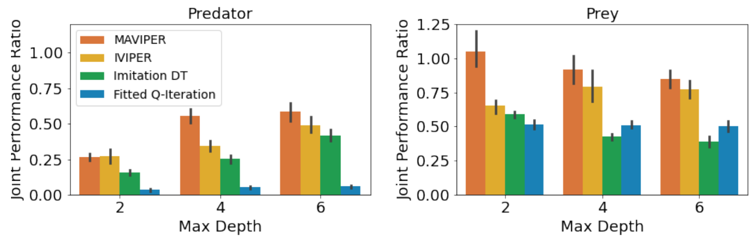
Furthermore, MAVIPER achieves a performance ratio above 0.75 in all environments with a maximum depth of 6. The same is true for IVIPER, except for the adversaries in physical deception. That means DT policies generated by IVIPER and MAVIPER lead to a performance degradation of less than or around 20% compared to the less interpretable NN-based expert policies. These results show that IVIPER and MAVIPER generate reasonable DT policies and outperform the baselines overall when adopted by a single agent.



(a) Physical Deception



(b) Cooperative Navigation



(c) Predator-prey

Fig. 3: Joint performance ratio: Relative performance when all agents in a team adopt DT policy and other agents use expert policy.

Joint Performance Compared to Experts A crucial aspect in multi-agent environments is agent coordination, especially when agents are on the same team with shared goals. To ensure that the DT policies capture this coordination, we analyze the performance of the DT policies when all agents in a team adopt DT policies, while other agents use expert policies. We define the joint performance ratio as: $\frac{U_Z(\hat{\pi}_Z, \pi^*_{-Z})}{U_Z(\pi^*)}$, where $U_Z(\hat{\pi}_Z, \pi^*_{-Z})$ is the utility of team Z when using their DT policies against the expert policies of the other agents $-Z$. Figure 3 shows the mean joint performance ratio for each team, averaged over all trials.

Figure 3a shows that MAVIPER defenders outperform IVIPER and the baselines, indicating that it better captures the coordinated behavior necessary to succeed in this environment. Fitted Q-Iteration struggles to achieve coordinated behavior, despite obtaining non-zero success for individual agents. This algorithm cannot capture the coordinated behavior, which we suspect is due to poor Q-value estimates. We hypothesize that the superior performance of MAVIPER is partially due to the defender agents correctly splitting their "attention" to the two targets to induce the correct behavior of covering both targets. To investigate this, we inspect the normalized average feature importances of the DT policies of depth 4 for both IVIPER and MAVIPER over 5 of the trials, as shown in Figure 4. Each of the MAVIPER defenders (top) most commonly focuses on the attributes associated with one of the targets. More specifically, defender 1 focuses on target 2 and defender 2 focuses on target 1. In contrast, both IVIPER defenders (bottom) mostly focus on the attributes associated with the goal tar-

Table 1: Robustness results. We report mean team performance and standard deviation of DT policies for each team, averaged across a variety of opponent policies. The best-performing algorithm for each agent type is shown in **bold**.

Environment	Team	MAVIPER	IVIPER	Imitation DT	Fitted Q-Iteration
Physical Deception	Defender	**.77** (.01)	.33 (.01)	.24 (.03)	.004 (.00)
	Adversary	**.42** (.03)	**.41** (.03)	**.42** (.03)	.07 (.01)
Predator-prey	Predator	**2.51** (0.72)	**1.98** (0.58)	1.14 (0.28)	0.26 (0.11)
	Prey	1.76 (0.80)	**2.16** (1.24)	**2.36** (1.90)	1.11 (0.82)

get. Not only does this overlap in feature space mean that defenders are unlikely to capture the correct covering behavior, but it also leaves them more vulnerable to an adversary, as it is easier to infer the correct target.

Figure 3b shows that MAVIPER agents significantly outperform all other algorithms in the cooperative navigation environment for all maximum depths. IVIPER agents significantly outperform the baselines for a maximum depth of 2 but achieve similar performance to the Imitation DT for the other maximum depths (where both algorithms significantly outperform the Fitted Q-Iteration baseline). MAVIPER better captures coordinated behavior, even as we increase the complexity of the problem by introducing another cooperating agent.

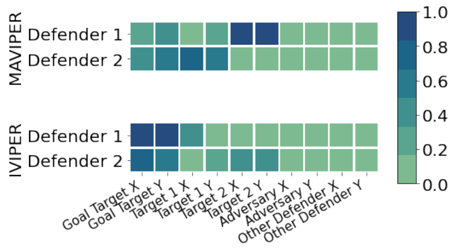


Fig. 4: Features used by the two defenders in the physical deception environment. Actual features are the relative positions of that agent and the labeled feature. Darker squares correspond to higher feature importance. MAVIPER defenders most commonly split importance across the two targets.

Figure 3c shows that the prey teams trained by IVIPER and MAVIPER outperform the baselines for all maximum depths. The predator teams trained by IVIPER and MAVIPER similarly outperform the baselines for all maximum depths. Also, MAVIPER leads to better performance than IVIPER in two of the settings (prey with depth 2 and predator with depth 4) while having no statistically significant advantage in other settings. Taken together, these results indicate that IVIPER and MAVIPER better capture the coordinated behavior necessary for a team to succeed in different environments, with MAVIPER significantly outperforming IVIPER in several environments.

Robustness to Different Opponents We investigate the robustness of the DT policies when a team using DT policies plays against a variety of oppo-

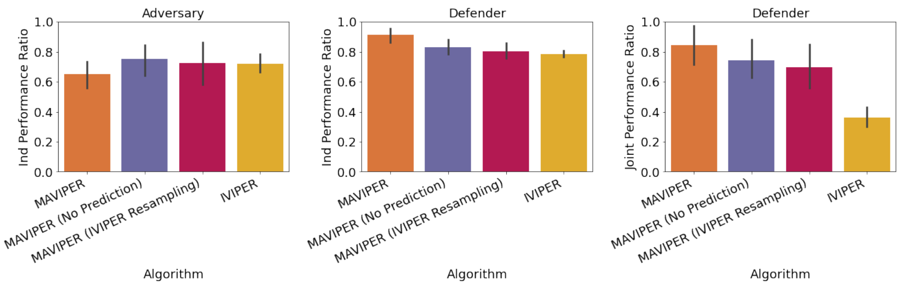


Fig. 5: Ablation study for MAVIPER for a maximum depth of 4. MAVIPER (No Prediction) does not utilize the predicted behavior of the anticipated DTs of the other agents to grow each agent's tree. MAVIPER (IVIPER Resampling) uses the same resampling method as IVIPER.

nents in the mixed competitive-cooperative environments. For this set of experiments, we choose a maximum depth of 4. Given a DT policy profile $\hat{\pi}$, a team Z's performance against an alternative policy file π' used by the opponents is: $U_Z(\hat{\pi}_Z, \pi'_{-Z})$. We consider a broad set of opponent policies π', including the policies generated by MAVIPER, IVIPER, Imitation DT, Fitted Q-Iteration, and MADDPG. We report the mean team performance averaged over all opponent policies in Table 1. See Tables 3 and 4 in Appendix C for the full results.

For physical deception, MAVIPER defenders outperform all other algorithms, with a gap of 0.44 between its performance and the next-best algorithm, IVIPER. This result indicates that MAVIPER learns coordinated defender policies that perform well against various adversaries. MAVIPER, IVIPER, and Imitation DT adversaries perform similarly on average, with a similar standard deviation, which supports the idea that the adversary's desired behavior is simple enough to capture with a less-sophisticated algorithm. For predator-prey, MAVIPER predators and prey outperform all other algorithms. The standard deviation of the performance of all algorithms is high due to this environment's complexity.

Ablation Study As discussed in Section 3.2, MAVIPER improves upon IVIPER with a few critical changes. First, we utilize the predicted behavior of the anticipated DTs of the other agents to grow each agent's tree. Second, we alter the resampling probability to incorporate the average Q-values over all actions for the other agents. To investigate the contribution of these changes to the performance, we run an ablation study with a maximum depth of 4 on the physical deception environment. We report both the mean independent and joint performance ratios for the defender team in Figure 5, comparing MAVIPER and IVIPER to two variants of MAVIPER without one of the two critical changes. Results show that both changes contributed to the improvement of MAVIPER over IVIPER, especially in the joint performance ratio.

5 Related Work

Most work on interpretable RL is in the single-agent setting [26]. We first discuss techniques that directly learn DT policies. CUSTARD [42] extends the action space of the MDP to contain actions for constructing a DT, i.e., choosing a feature to branch a node. Training an agent in the augmented MDP yields a DT policy for the original MDP while still enabling training using any function approximator, like NNs, during training. By redefining the MDP, the learning problem becomes more complex, which is problematic in multi-agent settings where the presence of other agents already complicates the learning problem. A few other works directly learn DT policies [9, 24, 44] for single-agent RL but not for the purpose of interpretability. Further, these works have custom learning algorithms and cannot utilize a high-performing NN policy to guide training.

VIPER [2] is considered to be a post-hoc DT-learning method [2]; however, we use it to produce *intrinsically* interpretable policies for deployment. MOET [45] extends VIPER by learning a mixture of DT policies trained on different regions of the state space. The resulting policy is a linear combination of multiple trees with non-axis-parallel partitions of the state. We find that the performance difference between VIPER and MOET is not significant enough to increase the complexity of the policy structure, which would sacrifice interpretability.

Despite increased interest in interpretable single-agent RL, interpretable MARL is less commonly explored. One line of work generates explanations from non-interpretable policies. Some work uses attention [14, 17, 29] to select and focus on critical factors that impact agents in the training process. Other work generates explanations as verbal explanations with predefined rules [47] or Shapley values [12]. The most similar line of work to ours [15] approximates non-interpretable MARL policies to interpretable ones using the framework of abstract argumentation. This work constructs argument preference graphs given manually-provided arguments. In contrast, our work does not need these manually-provided arguments for interpretability. Instead, we generate DT policies.

6 Discussion and Conclusion

We proposed IVIPER and MAVIPER, the first algorithms, to our knowledge, that train interpretable DT policies for MARL. We evaluated these algorithms on both cooperative and mixed competitive-cooperative environments. We showed that they can achieve individual performance of at least 75% of expert performance in most environment settings and over 90% in some of them, given a maximum tree depth of 6. We also empirically validated that MAVIPER effectively captures coordinated behavior by showing that teams of MAVIPER-trained agents outperform the agents trained by IVIPER and several baselines. We further showed that MAVIPER generally produces more robust agents than the other DT-learning algorithms.

Future work includes learning these high-quality DT policies from fewer samples, e.g., by using dataset distillation [46]. We also note that our algorithms can work in some environments where the experts and DTs are trained on different sets of features. Since DTs can be easier to learn with a simpler set of features, future work includes augmenting our algorithm with an automatic feature selection component that constructs simplified yet still interpretable features for training the DT policies.

Acknowledgements This material is based upon work supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate (NDSEG) Fellowship Program. This research was sponsored by the U.S. Army Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the funding agencies or government agencies. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

1. Abbeel, P., Ng, A.: Apprenticeship learning via inverse reinforcement learning. In: ICML (2004)
2. Bastani, O., et al.: Verifiable reinforcement learning via policy extraction. In: NeurIPS (2018)
3. Berner, C., et al.: Dota 2 with large scale deep reinforcement learning. arXiv preprint 1912.06680 (2019)
4. Bhalla, S., et al.: Deep multi agent reinforcement learning for autonomous driving. In: Canadian Conf. Artif. Intell. (2020)
5. Brittain, M., Wei, P.: Autonomous air traffic controller: A deep multi-agent reinforcement learning approach. arXiv preprint arXiv:1905.01303 (2019)
6. Buciluă, C., et al.: Model compression. In: KDD (2006)
7. Chen, Z., et al.: Relace: Reinforcement learning agent for counterfactual explanations of arbitrary predictive models. arXiv preprint arXiv:2110.11960 (2021)
8. Degris, T., et al.: Learning the structure of factored Markov decision processes in reinforcement learning problems. In: ICML (2006)
9. Ernst, D., et al.: Tree-based batch mode reinforcement learning. JMLR **6** (2005)
10. Foerster, J., et al.: Stabilising experience replay for deep multi-agent reinforcement learning. In: ICML (2017)
11. Foerster, J., et al.: Counterfactual multi-agent policy gradients. In: AAAI (2018)
12. Heuillet, A., et al.: Collective explainable ai: Explaining cooperative strategies and agent contribution in multiagent reinforcement learning with shapley values. IEEE Comput. Intell. Magazine **17** (2022)
13. Hinton, G., et al.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
14. Iqbal, S., Sha, F.: Actor-attention-critic for multi-agent reinforcement learning. In: ICML (2019)
15. Kazhdan, D., et al.: Marleme: A multi-agent reinforcement learning model extraction library. In: IJCNN (2020)

16. Li, S., et al.: Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In: AAAI (2019)
17. Li, W., et al.: Sparsemaac: Sparse attention for multi-agent reinforcement learning. In: Int. Conf. Database Syst. for Adv. Appl. (2019)
18. Lipton, Z.: The mythos of model interpretability. ACM Queue **16**(3) (2018)
19. Littman, M.: Markov games as a framework for multi-agent reinforcement learning. In: Mach. Learning (1994)
20. Lowe, R., et al.: Multi-agent actor-critic for mixed cooperative-competitive environments. arXiv preprint arXiv:1706.02275 (2017)
21. Luss, R., et al.: Local explanations for reinforcement learning. arXiv preprint arXiv:2202.03597 (2022)
22. Malialis, K., Kudenko, D.: Distributed response to network intrusions using multiagent reinforcement learning. Eng. Appl. Artif. Intell. (2015)
23. Matignon, L., et al.: Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. Knowledge Eng. Review **27**(1) (2012)
24. McCallum, R.: Reinforcement learning with selective perception and hidden state. PhD Thesis, Univ. Rochester, Dept. of Comp. Sci. (1997)
25. Meng, Z., et al.: Interpreting deep learning-based networking systems. In: Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (2020)
26. Milani, S., et al.: A survey of explainable reinforcement learning. arXiv preprint arXiv:2202.08434 (2022)
27. Mohanty, S., et al.: Flatland-rl: Multi-agent reinforcement learning on trains. arXiv preprint arXiv:2012.05893 (2020)
28. Molnar, C.: Interpretable Machine Learning (2019)
29. Motokawa, Y., Sugawara, T.: Mat-dqn: Toward interpretable multi-agent deep reinforcement learning for coordinated activities. In: ICANN (2021)
30. Oliehoek, F., et al.: Optimal and approximate q-value functions for decentralized pomdps. JAIR **32** (2008)
31. Paszke, A., et al.: Automatic differentiation in pytorch (2017)
32. Pyeatt, L.: Reinforcement learning with decision trees. In: Appl. Informatics (2003)
33. Pyeatt, L., Howe, A.: Decision tree function approximation in reinforcement learning. In: Int. Symp. on Adaptive Syst.: Evol. Comput. and Prob. Graphical Models (2001)
34. Quinlan, J.: Induction of decision trees. Mach. Learning (1986)
35. Rashid, T., et al.: Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In: ICML (2018)
36. Ross, S., et al.: A reduction of imitation learning and structured prediction to no-regret online learning. In: AISTATS (2011)
37. Roth, A., et al.: Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy. arXiv preprint arXiv:1907.01180 (2019)
38. Shapley, L.: Stochastic games. PNAS **39**(10) (1953)
39. Son, K., et al.: Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. arXiv preprint arXiv:1905.05408 (2019)
40. Strehl, A., et al.: Efficient structure learning in factored-state mdps. In: AAAI (2007)
41. Sunehag, P., et al.: Value-decomposition networks for cooperative multi-agent learning. arXiv preprint arXiv:1706.05296 (2017)

42. Topin, N., et al.: Iterative bounding mdps: Learning interpretable policies via non-interpretable methods. In: AAAI (2021)
43. Tuyls, K., et al.: Reinforcement learning in large state spaces. In: Robot Soccer World Cup (2002)
44. Uther, W., Veloso, M.: The lumberjack algorithm for learning linked decision forests. In: Int. Symp. Abstract., Reformulation, and Approx. (2000)
45. Vasic, M., et al.: Moët: Interpretable and verifiable reinforcement learning via mixture of expert trees. arXiv preprint arXiv:1906.06717 (2019)
46. Wang, T., et al.: Dataset distillation. arXiv preprint arXiv:1811.10959 (2018)
47. Wang, X., et al.: Explanation of reinforcement learning model in dynamic multi-agent system. arXiv preprint arXiv:2008.01508 (2020)
48. Yu, C., et al.: The surprising effectiveness of mappo in cooperative, multi-agent games. arXiv preprint arXiv:2103.01955 (2021)