Making Teams and Influencing Agents: Efficiently Coordinating Decision Trees for Interpretable Multi-Agent Reinforcement Learning

Rex Chen, Stephanie Milani, Zhicheng Zhang, Norman Sadeh, Fei Fang

School of Computer Science, Carnegie Mellon University
rexc@cmu.edu, smilani@andrew.cmu.edu, zczhang@cmu.edu, sadeh@cs.cmu.edu, feifang@cmu.edu

Abstract

Poor interpretability hinders the practical applicability of multi-agent reinforcement learning (MARL) policies. Deploying interpretable surrogates of uninterpretable policies enhances the safety and verifiability of MARL for real-world applications. However, if these surrogates are to interact directly with the environment within human supervisory frameworks, they must be both performant and computationally efficient. Prior work on interpretable MARL has either sacrificed performance for computational efficiency or computational efficiency for performance. To address this issue, we propose HYDRAVIPER, a decision tree-based interpretable MARL algorithm. HYDRAVIPER coordinates training between agents based on expected team performance, and adaptively allocates budgets for environment interaction to improve computational efficiency. Experiments on standard benchmark environments for multi-agent coordination and traffic signal control show that HYDRAVIPER matches the performance of state-of-the-art methods using a fraction of the runtime, and that it maintains a Pareto frontier of performance for different interaction budgets.

1 Introduction

Over the past decade, *multi-agent reinforcement learning* (MARL) algorithms have achieved state-of-the-art performance in various challenging board and video games (Silver et al. 2016; Vinyals et al. 2019). They have also found success in other sequential decision-making scenarios based on critical real-world domains, including robotics (Orr and Dutta 2023), cybersecurity (Panfili et al. 2018), and traffic signal control (TSC) (Chen et al. 2020). However, the real-world applicability of these algorithms is hampered by two key challenges. First, the deep neural network (NN) architectures that are needed to achieve good performance have thousands to millions of parameters. Second, the behaviour of RL agents is difficult to predict and verify due to its dependence on complex state spaces and long time horizons. Thus, human stakeholders understand and trust RL agents less than their simpler counterparts, even if RL yields superior performance (Siu et al. 2021). Conversely, more interpretable representations of policies can help stakeholders build appropriate levels of trust in RL agents (Druce, Harradon, and Tittle 2021; Zhang, Liao, and Bellamy 2020).

In applications where the safety and verifiability of RL policies is critical (Gilbert et al. 2023; Jayawardana, Landler, and Wu 2021), such as TSC, users may deploy interpretable surrogate policies instead of expert NN policies. Such surrogate policies should be *performant* — capable of achieving high returns. In MARL, coordinating the training of surrogates is critical for performance: if multiple surrogates are deployed simultaneously, they cannot assume that they are interacting with performant experts, as their performance may be influenced by other agents' suboptimal behaviour.

At the same time, surrogate policies should be *computationally efficient*; it should be possible to generate them with minimal environment interactions and runtime. Frequent interactions with the environment are impractical in domains such as robotics (Finn et al. 2017) and TSC (Zang et al. 2020), and simulators that have sufficient fidelity for real-world transferrability (Chen et al. 2023) are computationally intensive. In human-in-the-loop frameworks where users provide oversight to correct undesirable policy behaviour (Mandel et al. 2017), the ability to quickly iterate on surrogate policies is also critical (Wu et al. 2023). More complex models capable of stronger performance and coordination capabilities are less efficient (Milani et al. 2024).

Decision trees (DTs) are an attractive model class for interpretable RL due to their comprehensibility (Silva et al. 2020). They also enable the design of responsible AI systems, as their branching rules can be easily verified and constrained by human experts or automated processes (Blockeel et al. 2023). DTs lie at the core of the imitation learning framework VIPER (Bastani, Pu, and Solar-Lezama 2018), which has been applied to distil NN-based RL policies into DTs in domains such as TSC (Jayawardana, Landler, and Wu 2021), autonomous vehicles (Schmidt et al. 2021), and robotics (Roth et al. 2023). However, generalising VIPER to the MARL setting is challenging. Past work (Milani et al. 2022) introduced two multi-agent VIPER algorithms, IVIPER and MAVIPER, which are both impractical for deployment. IVIPER fails to coordinate the training of DTs, thus sacrificing performance; MAVIPER trains DTs in a coordinated but computationally inefficient manner.

To this end, we introduce HYDRAVIPER[1], an efficient method to extract coordinated DT policies for cooperative

[1]Code: https://github.com/lythronaxargestes/hydraviper-public

MARL. Our method makes three key algorithmic contributions: (1) HYDRAVIPER coordinates agent training by jointly resampling the training dataset for each team of cooperative agents. (2) When interacting with the environment to collect a training dataset, HYDRAVIPER adaptively collects critical trajectories closer to convergence. (3) When interacting with the environment for evaluation, HYDRAVIPER uses a multi-armed bandit-based evaluation strategy to identify promising sets of trained surrogates. Experiments demonstrate that HYDRAVIPER achieves our goal of balancing performance and computational efficiency. HYDRAVIPER also improves the applicability of DT-based interpretable MARL policies: users can exchange training time for performance by altering its environment interaction budgets, but its performance remains optimal at different budget levels. Lastly, HYDRAVIPER's efficiency on large environments can be improved while maintaining coordination by dividing the agent set into mutually influential teams.

2 Related Work

Interpretable Multi-Agent Learning Past methods for interpretable MARL have focused on using feature importance measures to construct saliency maps (Iqbal and Sha 2019; Heuillet, Couthouis, and Díaz-Rodríguez 2022; Liu, Zhu, and Chen 2023; Motokawa and Sugawara 2023), building logical structures (Kazhdan, Shams, and Lio 2020; Wang et al. 2021; Ji, Li, and Xiao 2023), and defining domain concepts (Zabounidis et al. 2023). Each of these categories of methods has limitations. Feature importances and saliency maps are visually clear, but only highlight aspects of the state space without showing how policies use them. Policies based on logical rules and concepts allow users to align the execution of these policies with domain knowledge, but require extensive feature engineering. By contrast, we learn simple policy representations grounded directly in the environment feature space.

Decision Trees for Reinforcement Learning Relative to deep NNs, shallow DT policy representations are intrinsically (Molnar 2019) and empirically (Silva et al. 2020) more comprehensible. One line of work in DT-based RL directly trains DT policies (Silva et al. 2020; Topin et al. 2020; Crespi et al. 2023; Liu et al. 2025) using relaxations amenable to direct optimisation. However, these methods suffer from training instability and performance degradation. Another line of work follows the *VIPER* framework (Bastani, Pu, and Solar-Lezama 2018), in which a surrogate DT is trained by imitation learning of a performant expert. Although VIPER has achieved success in single-agent settings (Schmidt et al. 2021; Roth et al. 2023; Jayawardana, Landler, and Wu 2021; Zhu, Yin, and Chen 2022), only two VIPER-based algorithms exist for multi-agent settings: *IVIPER* and *MAVIPER* (Milani et al. 2022). IVIPER independently trains DTs for each agent in a decentralised manner, enjoying computational efficiency at the cost of performance due to a lack of coordination. MAVIPER jointly trains DTs in a centralised manner to achieve coordination, but suffers from computational inefficiency. Neither algorithm balances performance and computational efficiency.

3 Background

Markov Games We model agent cooperation as a team-based *Markov game*. A Markov game for N agents consists of a set of states $\mathcal{S}$ with initial state distribution $\rho : \mathcal{S} \to [0, 1]$, and sets of actions $\mathcal{A}_1, ..., \mathcal{A}_N$ and observations $\mathcal{O}_1, ..., \mathcal{O}_N$ (consisting of features correlated with the state) for each agent i. The agent set is partitioned into disjoint teams $\mathcal{T}_1, \ldots, \mathcal{T}_L \subseteq [N]$. Each agent chooses actions according to a policy $\pi_i : \mathcal{O}_i \to \mathcal{A}_i$. After agents simultaneously execute actions $\mathbf{a}$, the environment produces the next state based on the state transition function $P : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \to \mathcal{S}$, a joint reward for each team $\mathcal{T}_\ell$ based on $R_\ell : \mathcal{S} \times \prod_{i \in \mathcal{T}_\ell} \mathcal{A}_i \to \mathbb{R}$, and private observations $O_i : \mathcal{S} \to \mathcal{O}_i$. Each agent maximises its team's return $\mathcal{R}_\ell = \sum_{t=0}^{T} \gamma^t R_\ell(s_t, \pi(\mathbf{o}_t))$ over T timesteps, where γ is a discount factor weighting the importance of future rewards.

MARL We refer to a policy profile as $\pi = (\pi_1, \ldots, \pi_N)$, a policy profile excluding agent i as π_{-i}, a joint policy profile for team $\mathcal{T}_\ell$ as $\pi_\ell = (\pi_i, \forall i \in \mathcal{T}_\ell)$, and a joint policy profile excluding all agents in $\mathcal{T}_\ell$ as $\pi_{-\ell} = (\pi_i, \forall i \notin \mathcal{T}_\ell)$. Each agent's *value function* and *state-action value* (or Q) *function* characterise its expected returns under a policy profile π:

$$V^{\pi_i}(s) = r_i + \gamma \sum_{s' \in \mathcal{S}} P(s, \pi_1(o_1), \ldots, \pi_N(o_N), s') V_i^{\pi}(s'),$$

$$Q^{\pi_i}(s, \mathbf{a}) = r_i + \gamma \sum_{s' \in S} P(s, a_1, \ldots, a_N, s') V^{\pi_i}(s')$$

Our algorithm assumes access to value and Q-functions that take the global observations of all agents, not states, as input: $V^{\pi_i}(\mathbf{o})$ and $Q^{\pi_i}(\mathbf{o}, \mathbf{a})$. Such Q-functions are used by actor-critic MARL algorithms that rely on centralised training with decentralised execution (Lowe et al. 2017; Foerster et al. 2018). Lastly, we define *mean* value functions and state-action value functions for each team: $\bar{V}^{\pi_\ell}(\mathbf{o}) := \frac{1}{|\mathcal{T}_\ell|} \sum_{i \in \mathcal{T}_\ell} V^{\pi_i}(\mathbf{o}), \bar{Q}^{\pi_\ell}(\mathbf{o}, \mathbf{a}) := \frac{1}{|\mathcal{T}_\ell|} \sum_{i \in \mathcal{T}_\ell} Q^{\pi_i}(\mathbf{o}, \mathbf{a})$.

Decision Trees A *decision tree* (DT) recursively partitions an input space $\mathcal{X}$ through functions $f_j : \mathcal{X} \to \mathbb{R}$ and thresholds τ_j at each internal node j. Each internal node induces a partition of samples, $P_j = x \in \mathcal{X} : f_j(x) \leq \tau_j$. For a DT policy, internal nodes (f_j, τ_j) encode observation-dependent decision criteria, while leaf nodes $l \in \mathcal{L}$ map partitioned observations to actions: $\hat{\pi}_i : \mathcal{O}_i \to \mathcal{A}_i, \forall i \in [N]$.

VIPER VIPER (Bastani, Pu, and Solar-Lezama 2018) is an imitation learning framework, adapted from the more general DAGGER (Ross, Gordon, and Bagnell 2011), that trains DTs as surrogate policies. Given a trained *expert* (NN) policy π^*, VIPER iteratively generates *student* (DT) policies $\hat{\pi}^m$. Specifically, in each iteration m, VIPER:

(1) *Collects* K new rollouts $\{\mathbf{o}, \hat{\pi}^{m-1}(\mathbf{o})\}$ using the previous students from iteration $m - 1$ (where $\hat{\pi}^0 := \pi^*$)

(2) *Resamples* a dataset $\mathcal{D}$ from all trajectories collected so far, based on upweighting critical states where taking a suboptimal action may be costly in terms of Q-values:

$$p_k \propto V^{\pi^*}(\mathbf{o}_k) - \min_{\mathbf{a}} Q^{\pi^*}(\mathbf{o}_k, \mathbf{a})$$

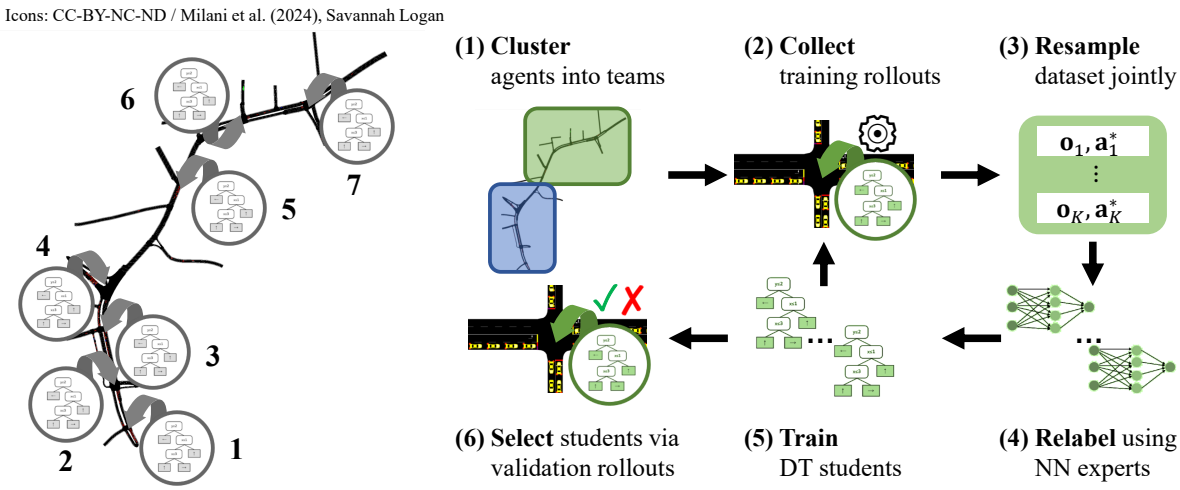


Figure 1: (L) Imitation learning in traffic signal control, where a decision tree must be learnt to imitate the RL-based policy of each intersection's signal controller agent (the seven-intersection Ingolstadt corridor TSC environment from Section 5.1 is shown, with intersections numbered). (R) The HYDRAVIPER framework, in which DT students are trained *independently* using a *jointly resampled* dataset of environment trajectories and relabelled by an NN expert.

(3) *Relabels* the dataset with the expert actions $\pi^*(\mathbf{o}_k)$

(4) *Trains* new DT students $\hat{\pi}^m$ on $\{\mathbf{o}_k, \pi^*(\mathbf{o}_k) \mid \mathbf{o}_k \in \mathcal{D}\}$

After M iterations, VIPER (5) *selects* a student through validation on an additional set of rollouts. Ross, Gordon, and Bagnell (2011) showed that such a procedure is guaranteed to find a student which is performant on the distribution of states that it induces.

4 HYDRAVIPER

In this section, we present *HYDRAVIPER* (Algorithm 1), our algorithm for performant and efficient interpretable MARL. As shown in Figure 1, HYDRAVIPER builds on the DAGGER and VIPER frameworks by iteratively collecting data from environment rollouts to train DT policies. HYDRAVIPER first (1) *partitions agents* into clusters for scalability (line 4). Next, in each of M iterations, HYDRAVIPER: (2) collects a dataset of rollouts from the environment, using an *adaptive procedure* (lines 6–7); (3) resamples the dataset to prioritise learning the correct actions in critical states, using *team-based Q-values* (lines 9–10); (4) and trains DTs based on these datasets (lines 11–12). After it completes all M training iterations, HYDRAVIPER (5) identifies the best-performing student for each agent, using a *multi-armed bandit algorithm*, and returns them as a policy profile (lines 13–14). Now, we describe each of these algorithm components in detail.

4.1 Dataset Resampling: Centralised-Q Weighting

VIPER-based algorithms include a dataset resampling step (Algorithm 1, lines 9–10) so that students can focus their learning on more critical states. At a high level, they construct a training dataset by computing sample weights on the aggregated dataset of environment rollouts, typically using some notion of value based on the expert Q-functions. Measuring value is straightforward in the single-agent setting, but — as we have mentioned — a key obstacle in multi-agent learning is efficient coordination among agents. To address this challenge, HYDRAVIPER induces coordination in the resampling step using a team-based notion of value (Algorithm 2), but trains DTs independently for each agent.

Specifically, HYDRAVIPER resamples the dataset $\mathcal{D}$ for DT construction based on weights $p_{\ell k}$, which represent the relative importance of each sample for each team of agents $\mathcal{T}_\ell$ (Algorithm 2, line 1). Past work computed this importance based on *individual* Q-functions, meaning that each agent must maintain its own dataset $\mathcal{D}_i$ and induce coordination through (typically computationally expensive) joint training procedures. By contrast, we propose an intuitive change: HYDRAVIPER uses the mean of the expert Q-functions within each *team* of coordinated agents, $\bar{Q}^{\pi^*_\ell} := \frac{1}{|\mathcal{T}_\ell|} \sum_{j \in \mathcal{T}_\ell} Q^{\pi^*_j}$, to prioritise samples according to their value to the team. Then, we compute the weights as the difference in value between the optimal joint team action and the worst-case joint team action. Intuitively, highly-weighted samples are those where coordinating on joint actions matters for performance. The weights are defined as:

$$p_{\ell k} \propto \bar{Q}^{\pi^*_\ell}(\mathbf{o}_k, \pi^*(\mathbf{o}_k)) - \min_{\mathbf{a}_\ell} \bar{Q}^{\pi^*_\ell}(\mathbf{o}_k, \mathbf{a}_\ell, \pi^*_{-\ell}(\mathbf{o}_{-\ell k}))$$
$$= \bar{V}^{\pi^*_\ell}(\mathbf{o}_k) - \min_{\mathbf{a}_\ell} \bar{Q}^{\pi^*_\ell}(\mathbf{o}_k, \mathbf{a}_\ell, \pi^*_{-\ell}(\mathbf{o}_{-\ell k})). \quad (1)$$

For further gains in sample efficiency, HYDRAVIPER does not compute $p_{\ell k}$ by enumerating joint actions over all agents in the environment. Instead, it only enumerates possible joint actions $\mathbf{a}_\ell$ over the *team* and uses expert actions $\pi^*_{-\ell}(\mathbf{o}_{-\ell})$ for the opponent agents. This novel resampling procedure eliminates the need for per-agent datasets in IVIPER and MAVIPER, allowing agents to prioritise the same critical states without computationally expensive joint training.

HYDRAVIPER uses each team's jointly sampled dataset

Algorithm 1 HYDRAVIPER

Input: Markov game $(\mathcal{S}, \mathcal{A}, P, R_i, O_i)$, experts π^*, expert Q-functions Q^{π^*}, per-iteration rollout count K_{train}, rollout budgets $(B_{\text{train}}, B_{\text{valid}})$, threshold ϵ, iteration count M, scaling factor c, agent distance function d
Output: Trained students $\hat{\pi}$

1: **Initialise** dataset $\mathcal{D} \leftarrow \emptyset$, policies $\hat{\pi}_i^0 \leftarrow \pi_i^*, \forall i \in N$
2: **Initialise** rollout count $n_{\text{train}} \leftarrow 0$
3: **Initialise** dropped rollout count $K_{\text{drop}} \leftarrow \infty$
▷ Section 4.4: Agent Clustering
4: **Cluster** agents $\mathcal{T}_1, \ldots, \mathcal{T}_L \leftarrow$ **Partition**(Γ, π^*, d)
5: **for** $m \in \{1, \ldots, M\}$ **do**
▷ Section 4.2: Training Rollouts
6: $\mathcal{D}, n_{\text{train}} \leftarrow$ **TR-A**$(\mathcal{D}, \hat{\pi}^{m-1}, m, K_{\text{train}}, B_{\text{train}}, K_{\text{drop}}, n_{\text{train}})$
7: **Reinitialise** dropped rollout count $K_{\text{drop}} \leftarrow \infty$
8: **for** each team $\mathcal{T}_\ell \in \{\mathcal{T}_1, \ldots, \mathcal{T}_L\}$ **do**
▷ Section 4.1: Dataset Resampling
9: $\mathcal{D}'_\ell, K'_{\text{drop}} \leftarrow$ **C-Q**$(\mathcal{D}_\ell, \mathcal{T}_\ell, \pi^*, Q^{\pi^*}, \epsilon)$
10: $K_{\text{drop}} \leftarrow \min(K_{\text{drop}}, K'_{\text{drop}})$
11: **for** each agent $i \in \mathcal{T}_\ell$ **do**
12: $\hat{\pi}_i^m \leftarrow$ **TrainDT**$(\mathcal{D}'_\ell)$
13: **for** each team $\mathcal{T}_\ell \in \{1, \ldots, L\}$ **do**
▷ Section 4.3: Validation Rollouts
14: $\hat{\pi}_i, \forall i \in \mathcal{T}_\ell \leftarrow$ **VR-UCB**$(\{\hat{\pi}_\ell^m\}_{m=1}^M, \mathcal{T}_\ell, B_{\text{valid}}, c)$
15: **return** $\hat{\pi} = (\hat{\pi}_1, \ldots, \hat{\pi}_N)$

$\mathcal{D}_\ell$ to independently train DTs for each agent i (Algorithm 1, lines 8–9). The DT $\hat{\pi}_i$ uses individual observations o_i to fit π_i^*'s actions in the dataset. Modifying the input dataset rather than the training procedure provides HYDRAVIPER with flexibility in the choice of DT learning algorithm. We use CART (Breiman et al. 1984), but more advanced models such as random forests or mixtures of DTs (Vasić et al. 2022) can also be used to improve performance.

4.2 Training Rollouts: Adaptive Budget Allocation

Thus far, we have assumed that HYDRAVIPER has access to a dataset of observation-action pairs for training. To collect this dataset, HYDRAVIPER follows the DAGGER-style iterative procedure of collecting a dataset at each iteration m by rolling out the current student policies $\hat{\pi}^{m-1}$ (Algorithm 1, line 6–7). The next set of students are trained on the aggregate of all collected datasets, therefore building up the set of inputs likely to be encountered by the student policies during execution. However, collecting training rollouts is computationally expensive. Past work has employed an inefficient static allocation strategy that uniformly performs K_{train} rollouts in each iteration. This strategy is problematic because the students are far from convergence early in training, so the distribution of trajectories collected earlier in training potentially diverges from those that converged students would encounter. HYDRAVIPER addresses this challenge through an adaptive rollout strategy that dynamically

Algorithm 2 Centralised-Q Resampling (C-Q)

Input: Team dataset $\mathcal{D}_\ell$, team $\mathcal{T}_\ell$, experts π^*, expert Q-functions Q^{π^*}, threshold ϵ
Output: Resampled dataset $\mathcal{D}'_\ell$, dropped rollout count K_{drop}

1: **Set** weights for each $(\mathbf{o}_k, \mathbf{a}_k) \in \mathcal{D}_\ell$:
$p_{\ell k} \leftarrow \bar{V}^{\pi^*_\ell}(\mathbf{o}_k) - \min_{\mathbf{a}_\ell} \bar{Q}^{\pi^*_\ell}(\mathbf{o}_k, \mathbf{a}_\ell, \pi^*_{-\ell}(\mathbf{o}_{-\ell k}))$
2: **Update** $K_{\text{drop}} \leftarrow \min_\ell \left\lceil \frac{1}{T} |\{(\mathbf{o}_k, \mathbf{a}_k) \in \mathcal{D}_\ell \mid p_{\ell k} \leq \epsilon\}| \right\rceil$
3: **Resample** dataset: $\mathcal{D}'_\ell \leftarrow \{(\mathbf{o}_k, \mathbf{a}_k) \sim p_{\ell k}\}$
4: **return** $\mathcal{D}'_\ell, K_{\text{drop}}$

Algorithm 3 Adaptive Training Rollouts (TR-A)

Input: Dataset $\mathcal{D}$, students $\hat{\pi}^{m-1}$, iteration m, per-iteration rollout count K_{train}, training rollout budget B_{train}, dropped rollout count K_{drop}, total rollout count n_{train}
Output: Updated dataset $\mathcal{D}$, total rollout count n_{train}

1: **Set** $K^m_{\text{train}} \leftarrow \min(K_{\text{drop}}, K_{\text{train}}) \mathbb{1}[n_{\text{train}} \leq B_{\text{train}}]$
2: **Update** $n_{\text{train}} \leftarrow n_{\text{train}} + K^m_{\text{train}} \mathbb{1}[m > 1]$
3: **for** each team $\mathcal{T}_\ell \in \{1, \ldots, L\}$ **do**
4: **Collect and relabel** K^m_{train} rollouts:
$\mathcal{D}^m_\ell \leftarrow \{(\mathbf{o}_\ell, \pi^*_\ell(\mathbf{o}_\ell)) \sim d^{(\hat{\pi}^{m-1})}\}$
5: **Aggregate** dataset: $\mathcal{D}_\ell \leftarrow \mathcal{D}_\ell \cup \mathcal{D}^m_\ell$
6: **return** $\mathcal{D}, n_{\text{train}}$

allocates the training budget at each iteration and prioritises critical states encountered later in training.

Recall that, for each team of cooperative agents $\mathcal{T}_\ell$, HYDRAVIPER follows Equation (1) to compute weights $p_{\ell k}$ for resampling the training dataset. We show the following:

Theorem 1. *Given a dataset of observation-action pairs for team $\mathcal{T}_\ell$ in iteration m of HYDRAVIPER, $\mathcal{D}_\ell = \{(\mathbf{o}_\ell, \mathbf{a}_\ell)\}$, assume there exists a pair $(\mathbf{o}_{\ell k}, \mathbf{a}_{\ell k})$ that receives the weight $p^{(m)}_{\ell k} = 0$. Then, in iteration $m+1$ of HYDRAVIPER, this pair also receives the weight $p^{(m+1)}_{\ell k} = 0$.*

Proof. See Appendix A.

As a result, samples $(\mathbf{o}_{\ell k}, \mathbf{a}_{\ell k})$ with $p_{\ell k} = 0$ are effectively *removed* from the dataset $\mathcal{D}$. This intuition serves as the motivation behind HYDRAVIPER's adaptive training rollout budget allocation (Algorithm 3): after samples are dropped during the resampling procedure, HYDRAVIPER performs rollouts to replenish the dataset.

Specifically, we treat the first iteration as a warm-up period, in which the experts collect a predefined number of K_{train} rollouts (Algorithm 3, lines 3–5). This leads to an initial dataset of $T \cdot K_{\text{train}}$ observation-action pairs. Each team $\mathcal{T}_\ell$ discards non-critical samples from its dataset (Algorithm 2, line 2), i.e. those where the range in the Q-value is at most a predefined threshold ϵ. With the goal of efficiency in mind, HYDRAVIPER computes the minimum number of such discarded samples across all teams of cooperative agents. This then determines the minimum number of rollouts required to collect at least this many samples in

Algorithm 4 UCB Validation Rollouts (VR-UCB)

Input: Policies $\{\hat{\pi}_\ell^m\}_{m=1}^M$, team $\mathcal{T}_\ell$, validation rollout budget B_{valid}, scaling factor c
Output: Selected policies $\hat{\pi}_i, \forall i \in \mathcal{T}_\ell$

1: **Initialise** $n_m \leftarrow 0$ for all $m \in \{1, \dots, M\}$
2: **Initialise** $n_{\min} \leftarrow \lceil 2 \ln B_{\text{valid}} \rceil$
3: **Initialise** return estimates: $\mu_\ell^m \leftarrow \frac{1}{C_{min}} \sum_{k=1}^{C_{min}} \bar{R}_{\ell k}, \bar{R}_{\ell k} \sim d^{(\hat{\pi}_\ell^m, \pi^*_{-\ell})}$
4: **for** rollout $k \in \{1, \dots, (B_{valid} - m n_{\min})\}$ **do**
5: **Set** $m^* \leftarrow \text{argmax}_m \hat{\mu}_\ell^m + \sqrt{\frac{c \ln B_{\text{valid}}}{n_m}}$
6: **Collect** mean return: $\bar{R}_{\ell k} \sim d^{(\hat{\pi}_\ell^{m^*}, \pi^*_{-\ell})}$
7: **Update** rollout count: $n_{m^*} \leftarrow n_{m^*} + 1$
8: **Update** running average of mean return: $\hat{\mu}_\ell^{m^*} \leftarrow \frac{n_{m^*}-1}{n_{m^*}} \hat{\mu}_\ell^{m^*} + \frac{1}{n_{m^*}} \bar{R}_{\ell k}$
9: **return** $\hat{\pi}_\ell^{m^*}, \forall i \in \mathcal{T}_\ell$

the next iteration. The expected number of dropped rollouts, and therefore the budget for the next iteration, is:

$$K_{\text{drop}} = \min_\ell \left\lceil \frac{1}{T} \left| \{(\mathbf{o}_k, \mathbf{a}_k) \in \mathcal{D}_\ell \mid p_{\ell k} \leq \epsilon\} \right| \right\rceil .$$

During the remaining $M-1$ iterations, HYDRAVIPER continues to collect rollouts using students until it exhausts its total budget of B_{train} training rollouts (Algorithm 3, line 1). Choosing different rollout budgets allows performance and efficiency to be traded off. A higher budget is likely to lead to superior performance, as more rollouts will be collected from students closer to convergence before the budget is exhausted, but it also requires more computation time.

4.3 Validation Rollouts: UCB Policy Selection

Following M iterations, HYDRAVIPER produces M joint policy profiles for each team. It then must select the best-performing policy profile (Algorithm 1, lines 13–14). HYDRAVIPER iterates through the policy profiles to estimate the team performance of each using a set of validation rollouts. The performance metric it uses is the undiscounted mean return of the team, $\bar{R}_\ell^m = \frac{1}{T} \sum_{t=0}^{T} R_\ell(s_t, \hat{\pi}_\ell^m(\mathbf{o}_t))$.

As is the case for training, collecting validation rollouts is computationally intensive, so these rollouts also need to be efficiently allocated. However, the problem setting differs here. Our goal is not to collect a *diverse* set of training rollouts, but rather to identify the *most performant* policy profiles using as few rollouts as possible. The mean return of each policy profile is unknown *a priori*; it must be estimated by selecting policy profiles and performing rollouts with noisy returns. Again, a fixed allocation strategy of K_{valid} environment rollouts for each policy profile is wasteful. The rollouts assigned to clearly poorly performing policy profiles could be reallocated to reduce the variance in the estimated returns of promising policy profiles. This motivation aligns with that of multi-armed bandit (MAB) problems.

Given a limited budget of B_{valid} rollouts, we represent the task of selecting the best-performing policy profile as a MAB problem. For each team $\mathcal{T}_\ell$, the policy profile $\hat{\pi}_\ell^m$ from each iteration m is an arm, and its return is a random variable $\bar{R}_\ell^m$ with unknown mean μ_ℓ^m. Each rollout samples from one such random variable, which captures the distribution of returns from environment and policy randomness. The objective is to identify the best arm $m_\ell^* = \text{argmax}_m \mu_\ell^m$ in as few rollouts as possible, i.e. to minimise the regret with respect to the policy that selects m_ℓ^* for every rollout.

In this work, we use a modification of the UCB1 algorithm (Auer, Cesa-Bianchi, and Fischer 2002). This allows us to achieve logarithmic regret given a readily satisfiable assumption: that the returns $\bar{R}_\ell^m$ of the arms are bounded (see Appendix B). Given a total budget of B_{valid} validation rollouts, HYDRAVIPER performs them as follows. For each policy profile, it first performs $n_{\min} = \lceil 2 \ln B_{\text{valid}} \rceil$ rollouts to generate initial estimates of the mean returns (Algorithm 4, lines 2–3). To allocate the remainder of the budget (lines 4–8), HYDRAVIPER follows UCB1 to select the policy profile index for the kth validation rollout as

$$m_{\ell k}^* = \underset{m}{\text{argmax}} \left(\hat{\mu}_\ell^m(k) + \sqrt{\frac{c \ln B_{\text{valid}}}{n_m(k)}} \right),$$

where $n_m(k) = \sum_{k'=1}^{k} \mathbb{1}[m_{\ell k'}^* = m]$ is the number of rollouts that have used policy profile m thus far, $\hat{\mu}_\ell^m(k) = \frac{\sum_{k'=1}^{k} \bar{R}_{\ell k'} \mathbb{1}[m_{\ell k'}^* = m]}{n_m(k)}$ is the empirical mean of the returns $\bar{R}_{\ell k}$ from policy profile m, B_{valid} is the total budget of rollouts, and c is a scaling constant for the confidence bound (see Section 5.4). HYDRAVIPER maintains a running average for the mean return of each policy profile, which it updates using the mean return $\bar{R}_{\ell k}$ of each rollout (line 8).

Algorithm 5 Agent Graph Clustering (Partition)

Input: Markov game $(\mathcal{S}, \mathcal{A}, P, R_i, O_i)$, experts π^*, agent distance function d
Output: Agent teams $\mathcal{T}_1, \dots, \mathcal{T}_L$

Construct graph $G = (V = \{1, \dots, N\}, E, w = 0)$
for each agent $i \in \{1, \dots, N\}$ **do**
  for each agent $j \in \{1, \dots, N\}$ **do**
    Assign edge weight $w_{ij} \leftarrow \frac{1}{d(i,j)}$
Partition graph $\mathcal{T}_1, \dots, \mathcal{T}_L \leftarrow$ **METIS**(G, L)
return $\mathcal{T}_1, \dots, \mathcal{T}_L$

4.4 Agent Clustering: Scaling Up HYDRAVIPER

When resampling the dataset, HYDRAVIPER calculates sample weights following Equation (1). This computation requires enumerating joint actions $\mathbf{a}_\ell$ for each team $\mathcal{T}_\ell$, in order to find the worst-case joint action that minimises the team's mean Q-function, $\min_{\mathbf{a}_\ell} \bar{Q}^{\pi_\ell^*}(\mathbf{o}_k, \mathbf{a}_\ell, \pi^*_{-\ell}(\mathbf{o}_{-\ell k}))$. The complexity of this step scales with the size of the joint action space and thus exponentially with the size of the team. Some mixed competitive-cooperative environments (see Section 5.1) have an inherent team structure that can reduce this complexity. In cooperative environments such as TSC, HYDRAVIPER clusters the agent set into teams to improve training efficiency (Algorithm 3).

Our goal is to find a clustering of the agent set into teams $\mathcal{T}_1 \dots \mathcal{T}_L$ so that HYDRAVIPER-trained DT students have *performance* similar to those trained on the full agent set, but improved *scalability* in that the number of actions to enumerate per team is much smaller than the full agent set: $\prod_{i \in \mathcal{T}_\ell} \mathcal{A}_i \ll \prod_{i \in \{1,\dots,N\}} \mathcal{A}_i$. We leverage the intuition that agents distant from each other (in terms of environmental distance, trajectory similarity, or other metrics) are unlikely to be influential on each other in most environments.

Suppose that we are given a function $d(i, j)$ that computes this distance between a pair of agents. In our clustering procedure (Algorithm 5), we first construct a complete graph $G = (V, E) = K_N$ where the nodes represent agents, and the weight between node i and node j, w_{ij}, is inversely proportional to $d(i, j)$ (lines 1–4). Then, we perform *graph partitioning* to divide G into L contiguous, connected node clusters of approximately equal size (line 5), such that the sum of the weights of inter-cluster edges is minimised. We use the hierarchical METIS algorithm (Karypis and Kumar 1998) to accomplish this. Note that we solve a graph partitioning problem instead of a min-cut problem to prevent the clusters from being imbalanced. Otherwise, in the worst case, the largest cluster could have size $O(N)$, thus yielding minimal gains in scalability.

How can the distance metric d be defined? For an environment that has an inherent team structure (such as the physical deception environment in Section 5.1), we define the graph G as a complete subgraph for each team. For traffic signal control environments, we note that the road network inherently forms a graph G_{env}, which can be partitioned to obtain sets of spatially proximal agents that correspond to neighbouring intersections. In this case, $G = G_{env}$.

Environment-agnostic distance metrics can also be designed. We consider a measure of proximity that aligns with the VIPER framework: the influence of agents on each other's Q-values. Recall from Section 4.1 that HYDRAVIPER measures the importance of samples by the mean Q-function $\bar{Q}^{\pi^*_\ell}$. In a cooperative setting, if one agent's actions have a significant impact on another agent's Q-values, then the joint actions of these agents are likely to have a significant impact on the overall agent set's mean Q-function. Including both agents on the same team would allow HYDRAVIPER to capture the effects of these joint actions. We define G using the distance function

$$d(i,j) = \mathbb{E}_{(\mathbf{o},\mathbf{a}) \in \mathcal{D}_{\text{dist}}} \frac{2}{\delta_{ij} + \delta_{ji}},$$

$$\delta_{ij} = Q^{\pi^*_j}(\mathbf{o}, \pi^*(\mathbf{o})) - \min_{a_i} Q^{\pi^*_j}(\mathbf{o}, a_i, \pi^*_{-i}(\mathbf{o}_{-i})),$$

where δ_{ij} is the range in agent j's Q-values induced by agent i, and the distance d_{ij} is the inverse of the average of δ_{ij} and δ_{ji}. This symmetrisation of influence is a simplifying assumption to obtain a single weight for each edge. Alternative distance metrics could be designed to better capture agent pairs where one is significantly more influential than the other. Since METIS requires integral edge weights, we rescale δ_{ij} to percentiles between $\min_{i,j} \delta_{ij}$ and $\max_{i,j} \delta_{ij}$.

5 Experiments

Now, we demonstrate the utility of HYDRAVIPER for interpretable MARL using experiments in various benchmark environments. In doing so, we perform a functionally grounded evaluation of interpretability (Doshi-Velez and Kim 2017), where we assess the quality of the generated DTs in terms of *performance* and *computational efficiency*. As the DTs would be used directly in place of NN-based policies in deployment, we consider these to be good proxy metrics for their practical applicability. More specifically, we address the following research questions:

RQ 1. *Is HYDRAVIPER both performant and efficient (in terms of environment interactions and runtime)?*

RQ 2. *Does HYDRAVIPER maintain performance optimality as the environment interaction budget decreases?*

RQ 3. *Can HYDRAVIPER maintain performance optimality while scalability is improved through agent clustering?*

5.1 Environments

We evaluate HYDRAVIPER in four environments: two environments in the *multi-agent particle world* (MPE) benchmark (Lowe et al. 2017), and two *traffic signal control* (TSC) environments in the RESCO benchmark (Ault and Sharon 2021). In MPE environments, agents must navigate in a 2D space to achieve a coordinated objective, making these environments ideal for assessing coordination capabilities.

Cooperative navigation (CN) In this environment, a team of three agents must coordinate to split up and cover three different targets while avoiding collisions with each other.

Physical deception (PD) In this environment, a team of two defender agents must cooperate to protect two targets from an adversary agent. One of the two targets is the "goal" of the adversary; this is not known to the adversary, which can only observe the positions of the targets and defenders. We train the two defender agents against an NN adversary.

In TSC environments, each agent controls a single intersection by selecting different signal phases; each phase allows vehicles from a subset of lanes to pass through the intersection. Both environments are based on real-world road corridors reproduced in the traffic simulator SUMO (Alvarez Lopez et al. 2018). To interface with the simulator, we use the OpenAI Gym-style wrapper `sumo-rl` (Alegre, Bazzan, and da Silva 2021). We focus on imitating experts for all agents as a team.

Cologne corridor (CC) (Uppoor and Fiore 2011) This environment simulates three signalised intersections in a corridor from the city of Cologne (Köln), Germany. It has a total volume of 4 494 vehicles in 7–8 am rush hour traffic.

Ingolstadt corridor (IC) (Lobo et al. 2020) This larger environment simulates seven signalised intersections in a corridor from the city of Ingolstadt, Germany. It has a lower total volume of 3 031 vehicles in 4–5 pm rush hour traffic.

5.2 Baselines and Setup

We compare HYDRAVIPER with IVIPER and MAVIPER, which represent the state of the art in interpretable multi-agent RL with DT surrogate policies. In addition, we compare with *expert* policies — MADDPG (Lowe et al. 2017) for MPE and MPLight (Chen et al. 2020) for TSC — and an additional baseline, *imitation DT*. Imitation DT does not use students to collect rollouts, nor does it perform dataset resampling; it collects the same number of training rollouts as the other algorithms and trains DTs on the collected dataset. As imitation DT performs worse than the other algorithms by a wide margin, we do not include it in Table 1 or Figure 3 but show its performance in Appendix C.

For MPE environments, we use a horizon of 25 timesteps per episode, and we trained MADDPG for 60 000 episodes as the expert for the DT students to imitate. For TSC environments, we use a horizon of 125 timesteps per episode (each timestep represents 20 seconds of simulation time), and we trained MPLight for 500 episodes as the expert. All imitation learning algorithms were run for 100 iterations to produce DTs with a maximum depth of 4. IVIPER and MAVIPER ran $K_{\text{train}} = K_{\text{valid}} = 50$ training and validation rollouts per iteration for MPE (including for the initial iteration where rollouts are collected by the experts), and 10 rollouts per iteration for TSC. Imitation DT ran the same number of training rollouts. We set these to equalise the number of environment interactions per iteration.

We repeated all experiments 10 times with different random seeds, and we report the mean and 95% confidence interval of the reward over 10 rollouts performed with the final student policy profiles generated from these runs. Most experiments were run in parallel on a server with 56 2.75GHz AMD EPYC 7453 processors and 252 GiB of RAM. For these experiments, we report the *number of rollouts collected*, not *runtime*s, as the rollout time is roughly constant. However, we also report runtimes for the execution of IVIPER, MAVIPER, and HYDRAVIPER on all four environments. For these experiments, we use the `kernprof` profiler (v4.1.3) to run them in sequence, with no other concurrent processes running except system routines. These experiments were run on another server with 8 4.2GHz Intel i7-7700K processors and 62 GiB of RAM.

5.3 Results

RQ 1 HYDRAVIPER achieves strong, coordinated performance in a computationally efficient manner. First, we assess HYDRAVIPER's performance as we vary it between two environment interaction budget levels, high (5 000 training/5 000 validation rollouts for MPE, 1 000 training/1 000 validation rollouts for TSC) and low (500 training/1 500 validation rollouts for MPE, 100 training/100 validation rollouts for TSC). As shown in Table 1, HYDRAVIPER students perform better than or comparable to students trained by the most performant DT baseline (MAVIPER for MPE, IVIPER for TSC) in all environments at both budget levels. HYDRAVIPER's performance is also better than or comparable to the NN experts for all environments except cooperative navigation, in which all DT-based algorithms cannot achieve expert-level performance. In physical deception, although neither MAVIPER nor HYDRAVIPER substantially outperforms IVIPER given the considerable stochasticity in the environment, HYDRAVIPER achieves a level of performance much closer to MAVIPER, while its training time is an order of magnitude shorter than MAVIPER.

In TSC environments, HYDRAVIPER is the best performing algorithm at both the high and low interaction budget levels. Notably, HYDRAVIPER at the high budget level substantially outperforms the expert on the Ingolstadt corridor. By contrast, MAVIPER fails to coordinate the intersection agents and is in general the worst-performing algorithm. HYDRAVIPER more than halves the runtime of both IVIPER and MAVIPER on both TSC environments.

In Figure 2, we show a decision tree generated by HYDRAVIPER for one agent in the Ingolstadt corridor at the low budget level (100 training/100 validation rollouts). Even with this limited environment interaction budget, HYDRAVIPER generates DTs that are not only performant, but also intuitively sensible. Each internal node of the DT compares the number of queueing vehicles for some turning movement to a threshold; the left branch includes all samples where there are fewer vehicles than the threshold, while the right branch includes all samples where there are more than the threshold. The root assesses traffic from the westbound side street. If it is low, the DT coordinates north-south traffic on the main road; if it is high, the DT coordinates turn traffic from the side street. Stakeholders, such as traffic engineers, can follow such a workflow to visualise, reason about, and supervise DTs generated by HYDRAVIPER.

RQ 2 As the environment interaction budget decreases, HYDRAVIPER still outperforms baselines. Now, we investigate the ability of HYDRAVIPER to adapt to increasing budget constraints for environment interaction, as would be imposed by users who wish to quickly iterate on DT policy training. As shown in Figure 3, HYDRAVIPER's performance in all four environments does not change substantially as the training and validation rollout budgets are individually reduced. Furthermore, in all four environments, HYDRAVIPER achieves performance on par with or better than MAVIPER at all budget levels. Therefore, HYDRAVIPER is able to maintain a Pareto frontier in the trade-off between performance and computational efficiency.

In cooperative navigation, the performance of both HYDRAVIPER and MAVIPER remains similar as the training and validation budgets are reduced individually. However, when both budgets are reduced simultaneously (shown in Figure 5), the performance of HYDRAVIPER but not MAVIPER remains essentially unchanged. In physical deception, HYDRAVIPER still performs well even as its training budget is reduced by a factor of 10, whereas MAVIPER performs substantially worse. Furthermore, the 95% confidence intervals of HYDRAVIPER's rewards are smaller than those of MAVIPER at all validation budget levels. Thus, HYDRAVIPER is able to identify performant policy profiles more consistently than MAVIPER.

In the Cologne corridor, HYDRAVIPER's performance consistently remains within the expert's 95% confidence in-

Environment		Expert	IVIPER	MAVIPER	HYDRAVIPER	HYDRAVIPER LB
Cooperative Navigation	*Total Penalty*	122.67 ± 1.67	160.87 ± 4.31	144.35 ± 2.12	144.48 ± 2.67	144.84 ± 2.12
	Runtime (s)	N/A	2 444.6 ± 9.1	21 188.7 ± 408.6	206.2 ± 11.1	180.5 ± 9.3
Physical Deception	*Defender Reward*	8.19 ± 0.50	6.94 ± 0.52	7.74 ± 0.82	7.72 ± 0.53	7.12 ± 0.84
	Runtime (s)	N/A	2 017.2 ± 21.3	11 782.4 ± 137.8	1 173.5 ± 21.6	388.4 ± 5.6
Cologne Corridor	*Queue Length*	18.94 ± 2.49	22.06 ± 2.91	25.85 ± 5.22	16.72 ± 1.51	18.77 ± 3.69
	Runtime (s)	N/A	33 841.4 ± 441.3	37 503.8 ± 834.8	13 651.6 ± 254.2	1 865.4 ± 26.1
Ingolstadt Corridor	*Queue Length*	23.01 ± 1.10	21.51 ± 2.13	24.26 ± 2.54	19.77 ± 1.51	21.87 ± 1.59
	Runtime (s)	N/A	75 927.5 ± 203.3	58 676.4 ± 1 392.3	11 263.8 ± 55.8	6 462.4 ± 30.9

Table 1: Performance and runtimes (means and 95% confidence intervals) for HYDRAVIPER and baselines. All algorithms are given the same environment interaction budget, except for low budget (LB) HYDRAVIPER (which uses 20% of the rollouts for MPE, 10% of the rollouts for TSC). HYDRAVIPER achieves or exceeds the performance of MAVIPER using a fraction of the runtime, and still performs well in the low budget setting. For physical deception, higher rewards are better; for all other environments, lower rewards are better. Appendix D shows runtimes for individual algorithm steps.

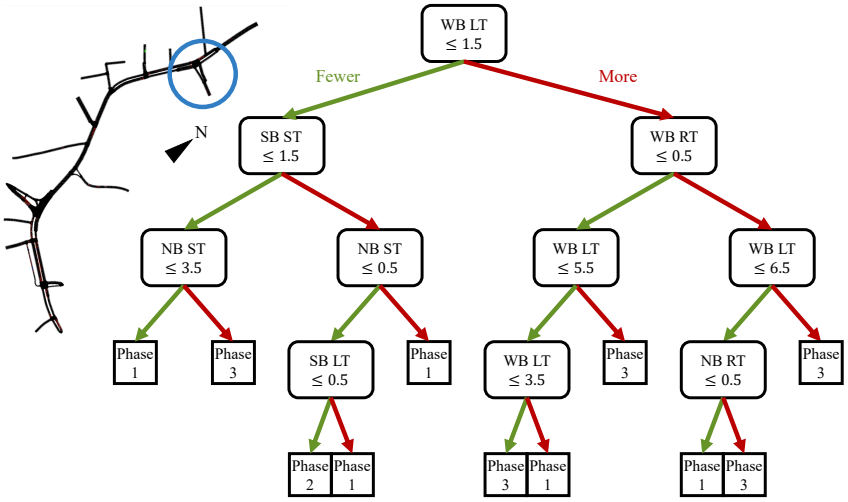


Figure 2: Decision tree generated by HYDRAVIPER (100 training/100 validation rollouts) for the Ingolstadt corridor (IC).

terval at all environment interaction budget levels, whereas the same is not true of MAVIPER. Meanwhile, the performance of HYDRAVIPER on the Ingolstadt corridor substantially exceeds the expert at all budget levels, whereas MAVIPER and IVIPER (except for the 500 validation rollout setting) remain in the expert's 95% confidence interval.

RQ 3 Even when the agent set is decomposed through clustering, HYDRAVIPER maintains its performance. Finally, we evaluate the effect of agent clustering on the performance of HYDRAVIPER. For the Ingolstadt corridor environment in the high budget setting (1 000 training/1 000 validation rollouts), we evaluate two strategies from Section 4.4: (1) clustering the agent set into two teams based on the road network graph G_{env} (`graph-metis`), and (2) using pairwise Q values to identify mutually impactful agents, and either k-means clustering (`marginal-kmeans`) or METIS (`marginal-metis`) for partitioning.

As shown in Figure 4, these clustering strategies allow HYDRAVIPER to retain its performance even when the size of the agent set is approximately halved for each team. Clustering reduces the largest team's joint action set in size from 1 944 to 72, and the total runtime of HYDRAVIPER by up to 47%. The best-performing strategy combines pairwise Q-value weights with METIS for partitioning, but they all perform similarly to the unpartitioned algorithm.

These clustering methods also outperform two baselines. First, the `random` baseline randomly assigns each agent to one of two teams; this baseline has very high variance in performance. Second, the `contiguous` baseline uses a handcrafted division of the agent set into two subsets; `graph-metis` recovers this division automatically, but `marginal-metis` improves further by grouping agents that are not adjacent in the environment.

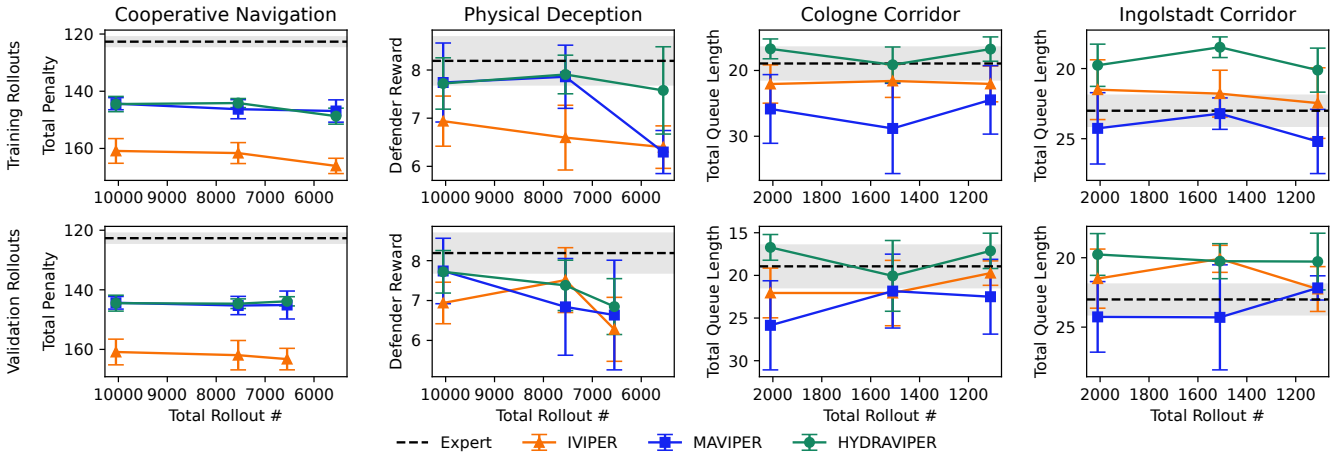
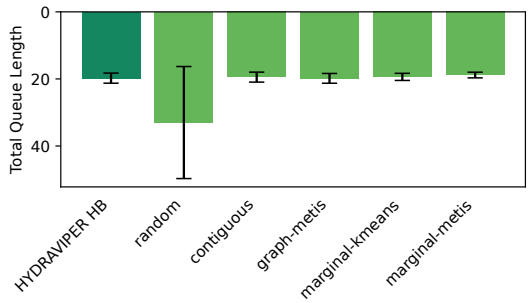


Figure 3: Performance of HYDRAVIPER and baselines as the number of rollouts decreases. Top shows decreasing training rollouts; bottom shows decreasing validation rollouts. HYDRAVIPER's performance stays consistent as the number of rollouts decreases. For physical deception, higher rewards are better; for all other environments, lower rewards are better. Bars show 95% confidence intervals based on 10 randomly-seeded runs. Full results are shown in Table 2 in Appendix C.



Clustering Method	Runtime (s)	Worst Clustering
HYDRAVIPER HB	11 263.8 ± 55.8	[[1,2,3,4,5,6,7]]
+ random	6 925.5 ± 897.0	[[1,2,3,6,7],[4,5]]
+ contiguous	6 731.3 ± 214.3	[[1,2,3],[4,5,6,7]]
+ graph-metis	6 026.4 ± 39.1	[[1,2,3],[4,5,6,7]]
+ marginal-kmeans	8 538.0 ± 643.0	[[1,2,3,4,6,7],[5]]
+ marginal-metis	6 067.5 ± 57.0	[[1,2,4,7],[3,5,6]]

Figure 4: (L) Performance of HYDRAVIPER on the Ingolstadt corridor (IC) under different agent set clustering methods at the high budget (HB) level. (R) Runtimes and worst-performing clusterings across 10 different random seeds of HYDRAVIPER under these methods. The intersection agent numbers follow those shown in Figure 1.

5.4 Hyperparameter Sensitivity

To understand the effects of HYDRAVIPER's hyperparameters on its performance, we conduct experiments to vary the depth of the DT students, and the scaling constant c for UCB policy selection (Section 4.3), on the cooperative navigation environment. We choose this environment due to its relatively low level of randomness. For these experiments, we use HYDRAVIPER at the low budget level (500 training/1 500 validation rollouts) as the baseline algorithm, and fix all hyperparameters other than those of interest. Figures for all results are shown in Appendix E.

By default, we use a DT depth of 4; our results show that DTs of this depth provide a good tradeoff between expressiveness and computational efficiency. Depth-4 DTs outperform depth-2 and depth-3 DTs on cooperative navigation. This is an intuitive result; the optimal agent policy for this environment cannot be represented with such shallow DTs, as they must condition on the positions of the other agents and the landmarks. However, we find that depth-4 DTs also marginally outperform depth-5 DTs. This same pattern exists in all of the environments that we use for evaluation. We hypothesise that the amount of data collected by HYDRAVIPER at the low budget level is insufficient to coordinate between depth-5 DTs.

Our default value for c is also 4. As discussed in Appendix B, this value is smaller than would be necessary according to a theoretical analysis. Since the agents are already trained, we hypothesise that the potential range of returns is less useful in practice for finding a good policy profile than the typical range of returns. The best DT depths for the other environments are all 4, as with cooperative navigation, while the best tested values of c for physical deception and the Cologne corridor are, respectively, 2 and 16.

5.5 Ablation

Lastly, to understand which components of HYDRAVIPER are responsible for its success, we conduct an ablation study for HYDRAVIPER at two budget levels in the cooperative navigation (CN) and Cologne corridor (CC) environments. For the high budget level, we use 5 000 training/5 000 val-

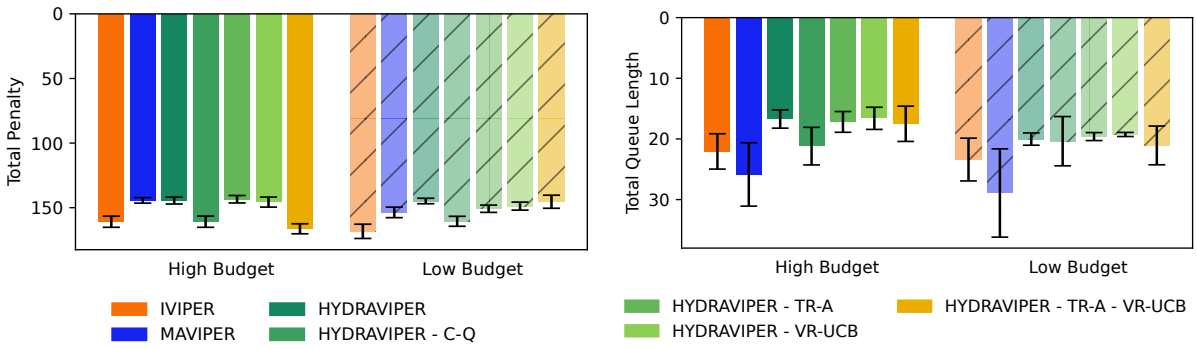


Figure 5: Ablation on cooperative navigation (CN, L) and Cologne corridor (CC, R). Lower rewards are better.

idation rollouts for CN, and 1 000 training/1 000 validation rollouts for CC; for the low budget level, we use 500 training/1 500 validation rollouts for CN, and 100 training/100 validation rollouts for CC. We compare HYDRAVIPER's centralised-Q resampling with IVIPER's independent resampling (HYDRAVIPER - CQ). In addition, we study the impact of removing adaptive training budget allocation (HYDRAVIPER - TR-A) and UCB-based validation budget allocation (HYDRAVIPER - VR-UCB). Results for physical deception and the Ingolstadt corridor are shown in Appendix F.

Figure 5 shows our ablation results. In both environments and at both budget levels, centralised-Q resampling outperforms the IVIPER resampling scheme, although the performance gap is less pronounced for the Cologne corridor due to environmental randomness. This result suggests that sampling the training dataset independently for each agent, instead of according to team performance, is insufficient to achieve coordinated behaviour in the resulting students. Meanwhile, removing the budget allocation methods degrades the performance of HYDRAVIPER. Having either one of the budget allocation methods is generally sufficient to improve HYDRAVIPER's reward, except in one case: for cooperative navigation at the low budget level, HYDRAVIPER performs worse when only one budget allocation mechanism is present. Meanwhile, for the Cologne corridor at the low budget level, the variance in HYDRAVIPER's reward is large both when *only* centralised-Q resampling is present, and also when it is *removed*. These results suggest that the primary benefit of the two rollout budget allocation mechanisms is to stabilise HYDRAVIPER's learning process, especially in the low budget setting when extracting the most information from each rollout is critical.

6 Conclusion and Future Work

In this work, we introduced a new DT-based interpretable MARL method, HYDRAVIPER. HYDRAVIPER addresses several limitations of prior multi-agent methods that follow the VIPER framework: (1) it improves performance by using a joint dataset resampling scheme based on team Q-values, and (2) it improves computational efficiency by adaptively allocating fixed budgets of environment interactions for training and validation, as well as by dividing agents into jointly-trained teams. Based on experiments in benchmark environments for multi-agent coordination and traffic signal control, we showed that HYDRAVIPER achieves performance comparable with MAVIPER (a centralised method) and even neural network experts, all with a runtime less than IVIPER (a decentralised method). We also demonstrated HYDRAVIPER's sample efficiency in its ability to retain a similar level of performance using a fraction of the environment interactions.

Through our experiments in the Ingolstadt corridor environment, we scaled up the VIPER framework to seven agents. To our knowledge, this is the largest team of coordinated agents to which interpretable MARL has been applied so far. However, environments based on real-world domains can have many more agents than the environments that we studied. For example, the review of Noaeen et al. (2022) showed that TSC environments of dozens or even hundreds of agents are used in the RL literature. In the most extreme case, Chen et al. (2020) used parameter-shared MPLight policies as controller agents for an extremely large simulation of 2 510 traffic lights. Our agent clustering approach shows promise in scaling up to larger environments while retaining performance comparable to that of expert policies. We envision that the flexibility of the HYDRAVIPER framework will allow it to adapt to characteristics of different MARL environments while maintaining Pareto optimality in the performance-computational efficiency tradeoff.

However, our functionally grounded evaluation has not shown whether the resulting DTs are sufficient to help stakeholders better understand these policies. An application-grounded evaluation (Doshi-Velez and Kim 2017) of HYDRAVIPER that includes user studies would be necessary to assess its practical utility. Combining the general framework of HYDRAVIPER with alternative policy structures, including those based on natural language, may help make these DT-based policies more understandable and controllable.

Acknowledgments

This work was supported by the Tang Family Endowed Innovation Fund and NSF grant IIS-2046640 (CAREER). Additionally, we thank Naveen Raman, Yixuan Xu, Jingwu Tang, Matteo Pozzi, and Peter Stone for their feedback.

References

Alegre, L. N.; Bazzan, A. L.; and da Silva, B. C. 2021. Quantifying the impact of non-stationarity in reinforcement learning-based traffic signal control. *PeerJ Computer Science*, 7: e575.

Alvarez Lopez, P.; Behrisch, M.; Bieker-Walz, L.; Erdmann, J.; Flötteröd, Y.-P.; Hilbrich, R.; Lücken, L.; Rummel, J.; Wagner, P.; and Wießner, E. 2018. Microscopic Traffic Simulation using SUMO. In *Proceedings of the 21st International Conference on Intelligent Transportation Systems*, ITSC '18, 2575–2582. Piscataway, USA.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47: 235–256.

Ault, J.; and Sharon, G. 2021. Reinforcement learning benchmarks for traffic signal control. In *Proceedings of the 35th Conference on Neural Information Processing Systems, Datasets and Benchmarks Track*, NeurIPS '21, 1–11. Virtual.

Bastani, O.; Pu, Y.; and Solar-Lezama, A. 2018. Verifiable Reinforcement Learning via Policy Extraction. In *Proceedings of the 32nd Conference on Neural Information Processing Systems*, NeurIPS '18, 2494–2504. Montréal, Canada.

Blockeel, H.; Devos, L.; Frénay, B.; Nanfack, G.; and Nijssen, S. 2023. Decision trees: from efficient prediction to responsible AI. *Frontiers in Artificial Intelligence*, 6: 1124553.

Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. J. 1984. Splitting Rules. In *Classification and Regression Trees*. New York: Taylor & Francis.

Chen, C.; Wei, H.; Xu, N.; Zheng, G.; Yang, M.; Xiong, Y.; Xu, K.; and Li, Z. 2020. Toward A Thousand Lights: Decentralized Deep Reinforcement Learning for Large-Scale Traffic Signal Control. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, AAAI '20, 3414–3421. New York, USA.

Chen, R.; Carley, K. M.; Fang, F.; and Sadeh, N. 2023. Purpose in the Machine: Do Traffic Simulators Produce Distributionally Equivalent Outcomes for Reinforcement Learning Applications? In *Proceedings of the 2023 Winter Simulation Conference*, WSC '23, 1842–1853. San Antonio, USA.

Crespi, M.; Ferigo, A.; Custode, L. L.; and Iacca, G. 2023. A population-based approach for multi-agent interpretable reinforcement learning. *Applied Soft Computing*, 147: 110758.

Doshi-Velez, F.; and Kim, B. 2017. Towards A Rigorous Science of Interpretable Machine Learning. *arXiv preprint arXiv:1702.08608*.

Druce, J.; Harradon, M.; and Tittle, J. 2021. Explainable artificial intelligence (XAI) for increasing user trust in deep reinforcement learning driven autonomous systems. *arXiv preprint arXiv:2106.03775*.

Finn, C.; Yu, T.; Zhang, T.; Abbeel, P.; and Levine, S. 2017. One-Shot Visual Imitation Learning via Meta-Learning. In *Proceedings of the 1st Annual Conference on Robot Learning*, CoRL '17, 357–368. Mountain View, USA.

Foerster, J.; Farquhar, G.; Afouras, T.; Nardelli, N.; and Whiteson, S. 2018. Counterfactual multi-agent policy gradients. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, AAAI '18, 2974–2982. New Orleans, USA.

Gilbert, T. K.; Lambert, N.; Dean, S.; Zick, T.; Snoswell, A.; and Mehta, S. 2023. Reward Reports for Reinforcement Learning. In *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '23, 84–130. Montréal, Canada.

Heuillet, A.; Couthouis, F.; and Díaz-Rodríguez, N. 2022. Collective eXplainable AI: Explaining Cooperative Strategies and Agent Contribution in Multiagent Reinforcement Learning With Shapley Values. *IEEE Computational Intelligence Magazine*, 17(1): 59–71.

Hoeffding, W. 1963. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58: 13–30.

Iqbal, S.; and Sha, F. 2019. Actor-Attention-Critic for Multi-Agent Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning*, ICML '19, 2961–2970. Long Beach, USA.

Jayawardana, V.; Landler, A.; and Wu, C. 2021. Mixed Autonomous Supervision in Traffic Signal Control. In *Proceedings of the 2021 IEEE 24th International Conference on Intelligent Transportation Systems*, ITSC '21, 1767–1773. Indianapolis, USA.

Ji, B.; Li, G.; and Xiao, G. 2023. Enhancing the Interpretability of Deep Multi-agent Reinforcement Learning via Neural Logic Reasoning. In *ICANN '23*, 199–210. Crete, Greece.

Karypis, G.; and Kumar, V. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1): 359–392.

Kazhdan, D.; Shams, Z.; and Lio, P. 2020. MARLeME: A Multi-Agent Reinforcement Learning Model Extraction Library. In *Proceedings of the 2020 International Joint Conference on Neural Networks*, IJCNN '20, 1–8. Glasgow, UK.

Liu, Z.; Zhu, Y.; and Chen, C. 2023. NA2Q: Neural Attention Additive Model for Interpretable Multi-Agent Q-Learning. In *Proceedings of the 40th International Conference on Machine Learning*, ICML '23, 22539–22558. Honolulu, USA.

Liu, Z.; Zhu, Y.; Wang, Z.; Gao, Y.; and Chen, C. 2025. MIXRTs: Toward Interpretable Multi-Agent Reinforcement Learning via Mixing Recurrent Soft Decision Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(5): 4090–4107.

Lobo, S.; Neumeier, S.; Fernandez, E. M. G.; and Facchi, C. 2020. InTAS - The Ingolstadt Traffic Scenario for SUMO. In *Proceedings of the 2020 SUMO User Conference*, SUMO '20, 73–92. Berlin, Germany.

Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, P.; and Mordatch, I. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, NeurIPS '17, 6379–6390. Long Beach, USA.

Mandel, T.; Liu, Y.-E.; Brunskill, E.; and Popović, Z. 2017. Where to Add Actions in Human-in-the-Loop Reinforcement Learning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, AAAI '17, 2322–2328. San Francisco, USA.

Milani, S.; Topin, N.; Veloso, M.; and Fang, F. 2024. Explainable Reinforcement Learning: A Survey and Comparative Review. *ACM Computing Surveys*, 56(7): 1–36.

Milani, S.; Zhang, Z.; Topin, N.; Shi, Z. R.; Kamhoua, C.; Papalexakis, E. E.; and Fang, F. 2022. MAVIPER: Learning Decision Tree Policies for Interpretable Multi-agent Reinforcement Learning. In *Proceedings of the 2022 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, ECML PKDD '22, 251–266. Grenoble, France.

Molnar, C. 2019. Taxonomy of Interpretability Methods. In *Interpretable Machine Learning*. Independent.

Motokawa, Y.; and Sugawara, T. 2023. Interpretability for Conditional Coordinated Behavior in Multi-Agent Reinforcement Learning. In *Proceedings of the 2023 International Joint Conference on Neural Networks*, IJCNN '23, 1–8. Gold Coast, Australia.

Noaeen, M.; Naik, A.; Goodman, L.; Crebo, J.; Abrar, T.; Abad, Z. S. H.; Bazzan, A. L.; and Far, B. 2022. Reinforcement learning in urban network traffic signal control: A systematic literature review. *Expert Systems with Applications*, 199: 116830.

Orr, J.; and Dutta, A. 2023. Multi-agent deep reinforcement learning for multi-robot applications: A survey. *Sensors*, 23(7): 3625.

Panfili, M.; Giuseppi, A.; Fiaschetti, A.; Al-Jibreen, H. B.; Pietrabissa, A.; and Priscoli, F. D. 2018. A game-theoretical approach to cyber-security of critical infrastructures based on multi-agent reinforcement learning. In *Proceedings of the 26th Mediterranean Conference on Control and Automation*, MED '16, 460–465. Zadar, Croatia.

Ross, S.; Gordon, G. J.; and Bagnell, J. A. 2011. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, AISTATS '11, 627–635. Fort Lauderdale, USA.

Roth, A. M.; Liang, J.; Sriram, R.; Tabassi, E.; and Manocha, D. 2023. MSVIPER: Improved Policy Distillation for Reinforcement-Learning-Based Robot Navigation. *Journal of the Washington Academy of Sciences*, 109(2): 27–58.

Schmidt, L. M.; Kontes, G.; Plinge, A.; and Mutschler, C. 2021. Can you trust your autonomous car? Interpretable and verifiably safe reinforcement learning. In *Proceedings of the 2021 IEEE Intelligent Vehicles Symposium*, IV '21, 171–178. Nagoya, Japan.

Silva, A.; Killian, T.; Jimenez, I. R.; Son, S.-H.; and Gombolay, M. 2020. Optimization Methods for Interpretable Differentiable Decision Trees in Reinforcement Learning. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, AISTATS '20, 1855–1865. Palermo, Italy.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529: 484–489.

Siu, H. C.; Peña, J.; Chen, E.; Zhou, Y.; Lopez, V.; Palko, K.; Chang, K.; and Allen, R. 2021. Evaluation of human-AI teams for learned and rule-based agents in Hanabi. In *Proceedings of the 35th Conference on Neural Information Processing Systems*, NeurIPS '21, 16183–16195. Virtual.

Topin, N.; Milani, S.; Fang, F.; and Veloso, M. 2020. Iterative Bounding MDPs: Learning Interpretable Policies via Non-Interpretable Methods. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, AAAI '20, 9923–9931. New York, USA.

Uppoor, S.; and Fiore, M. 2011. Large-scale urban vehicular mobility for networking research. In *Proceedings of the 2011 IEEE Vehicular Networking Conference*, VNC '11, 62–69. Amsterdam, Netherlands.

Vasić, M.; Petrović, A.; Wang, K.; Nikolić, M.; Singh, R.; and Khurshid, S. 2022. MoËT: Mixture of Expert Trees and its application to verifiable reinforcement learning. *Neural Networks*, 151: 34–47.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575: 350–354.

Wang, X.; Li, H.; Zhang, H.; Lewis, M.; and Sycara, K. 2021. Explanation of Reinforcement Learning Model in Dynamic Multi-Agent System. *arXiv preprint arXiv:2008.01508*.

Wu, J.; Huang, Z.; Hu, Z.; and Lv, C. 2023. Toward Human-in-the-Loop AI: Enhancing Deep Reinforcement Learning via Real-Time Human Guidance for Autonomous Driving. *Engineering*, 21: 75–91.

Zabounidis, R.; Campbell, J.; Stepputtis, S.; Hughes, D.; and Sycara, K. P. 2023. Concept Learning for Interpretable Multi-Agent Reinforcement Learning. In *Proceedings of the 7th Annual Conference on Robot Learning*, CoRL '23, 1828–1837. Atlanta, USA.

Zang, X.; Yao, H.; Zheng, G.; Xu, N.; Xu, K.; and Li, Z. 2020. MetaLight: Value-Based Meta-Reinforcement Learning for Traffic Signal Control. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, AAAI '20, 1153–1160. New York, USA.

Zhang, Y.; Liao, Q. V.; and Bellamy, R. K. E. 2020. Effect of confidence and explanation on accuracy and trust calibration in AI-assisted decision making. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, FAccT '20, 295–305. Barcelona, Spain.

Zhu, Y.; Yin, X.; and Chen, C. 2022. Extracting decision tree from trained deep reinforcement learning in traffic signal control. *IEEE Transactions on Computational Social Systems*, 10(4): 1997–2007.