RETHINKING THE IMPLEMENTATION TRICKS AND MONOTONICITY CONSTRAINT IN COOPERATIVE MULTI-AGENT REINFORCEMENT LEARNING

Jian Hu [*†]
Graduate Institute of Networking and Multimedia
National Taiwan University
Taipei
janhu9527@gmail.com

Siyang Jiang [*]
Graduate Institute of Electrical Engineering
National Taiwan University
Taipei
syjiang@arbor.ee.ntu.edu.tw

Seth Austin Harding
Department of Computer Science
National Taiwan University
Taipei
b06902101@ntu.edu.tw

Haibin Wu
Graduate Institute of Communication Engineering
National Taiwan University
Taipei
f07921092@ntu.edu.tw

Shih-wei Liao
Department of Computer Science
National Taiwan University
Taipei
liao@csie.ntu.edu.tw

ABSTRACT

Many complex multi-agent systems such as robot swarms control and autonomous vehicle coordination can be modeled as Multi-Agent Reinforcement Learning (MARL) tasks. QMIX, a widely popular MARL algorithm, has been used as a baseline for the benchmark environments, e.g., Starcraft Multi-Agent Challenge (SMAC), Difficulty-Enhanced Predator-Prey (DEPP). Recent variants of QMIX target relaxing the monotonicity constraint of QMIX, allowing for performance improvement in SMAC. In this paper, we investigate the code-level optimizations of these variants and the monotonicity constraint. (1) We find that such improvements of the variants are significantly affected by various code-level optimizations. (2) The experiment results show that QMIX with normalized optimizations outperforms other works in SMAC; (3) beyond the common wisdom from these works, the monotonicity constraint can improve sample efficiency in SMAC and DEPP. We also discuss why monotonicity constraints work well in purely cooperative tasks with a theoretical analysis. We open-source the code at https://github.com/hijkzzz/pymarl2.

1 INTRODUCTION

Multi-agent cooperative games have many complex real-world applications such as, robot swarm control [7; 36; 15], autonomous vehicle coordination [3; 40], and sensor networks [38], a complex task always requires multi-agents to accomplish together. Multi-Agent Reinforcement Learning (MARL), is used to solve the multi-agent systems tasks [36].

In multi-agent systems, a typical challenge is a limited scalability and inherent constraints on agent observability and communication. Therefore, decentralized policies that act only on their local observations are necessitated and widely used [39]. Learning decentralized policies is an intuitive

---

[*]Jian Hu and Siyang Jiang contributed equally to this work.
[†]Corresponding author.

approach for training agents independently. However, simultaneous exploration by multiple agents often results in non-stationary environments, which leads to unstable learning. Therefore, *Centralized Training and Decentralized Execution* (CTDE) [10] allows for independent agents to access additional state information that is unavailable during policy inference.

Many CTDE learning algorithms have been proposed for the better sample efficiency in cooperative tasks[35]. Among them, several value-based approaches achieve state-of-the-art (SOTA) performance [20; 32; 37; 21] on such benchmark environments, e.g., Starcraft Multi-Agent Challenge (SMAC) [22], Predator-Prey (PP) [2; 17]. To enable effective CTDE for multi-agent Q-learning, the Individual-Global-Max (IGM) principle [24] of equivalence of joint greedy action and individual greedy actions is critical. The primary advantage of the IGM principle is that it ensures consistency of policy with centralized training and decentralized execution. To ensure IGM principle, QMIX [20] was proposed for factorizing the joint action-value function with the *Monotonicity Constraint* [32], however, limiting the expressive power of the mixing network.

To improve the performance of QMIX, some variants of QMIX [1], including value-based approaches [37; 21; 32; 25] and a policy-based approach [39], have been proposed with the aim to relax the monotonicity constraint of QMIX. However, while investigating the codes of these variants, we find that their performance is significantly affected by their code-level optimizations (or implementation tricks). Therefore, it is left unclear whether monotonicity constraint indeed impairs the QMIX's performance.

In this paper, we investigate the impact of the *code-level optimizations* and the *monotonicity constraint* in cooperative MARL. Firstly, we investigate the effects of code-level optimizations, which enable QMIX to solve the most difficult challenges in SMAC. Afterward, we normalize the optimizations of QMIX and its variants; specifically, we perform the same hyperparameter search pattern for all algorithms, which includes using or removing a certain optimization and a grid hyperparameter search; the experiment results (Sec. 5.2.1) demonstrate that QMIX outperforms the other variants. Secondly, to study the impact of the monotonicity constraint, we propose a policy-based algorithm, RIIT; the experimental results (Sec. 5.2.2) show that the monotonicity constraint improves sample efficiency in SMAC and DEPP. Lastly, to generalize cooperative tasks beyond SMAC and DEPP, we give a strict definition of purely cooperative tasks and a discussion about why monotonicity constraints work well in purely cooperative tasks.

To our best knowledge, this work is the first to analyze the monotonicity constraint and code-level optimizations in MARL. Our analysis shows that QMIX works well if a multi-agent task can be interpreted as purely cooperative, even if it can also be interpreted as competitive.

2 Preliminaries

Dec-POMDP. We model the multi-robot RL problem as decentralized partially observable Markov decision process (Dec-POMDP) [16], which composed of a tuple $G = \langle \mathcal{S}, \mathcal{U}, P, r, \mathcal{Z}, O, N, \gamma \rangle$. $s \in S$ describes the true state of the environment. At each time step, each agent $i \in \mathcal{N} := \{1, \ldots, N\}$ chooses an action $u^i \in \mathcal{U}$, forming a joint action $\boldsymbol{u} \in \mathcal{U}^N$. All state transition dynamics are defined by function $P\left(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{u}\right) : \mathcal{S} \times \mathcal{U}^N \times \mathcal{S} \mapsto [0, 1]$. Each agent has independent observation $z \in \mathcal{Z}$, determined by observation function $O(s, i) : \mathcal{S} \times \mathcal{N} \mapsto \mathcal{Z}$. All agents share the same reward function $r(s, \boldsymbol{u}) : \mathcal{S} \times \mathcal{U}^N \to \mathbb{R}$ and $\gamma \in [0, 1)$ is the discount factor. The objective function, shown in Eq. 1, is to maximize the joint value function to find a joint policy $\boldsymbol{\pi} = \langle \pi_1, ..., \pi_n \rangle$.

$$J\left(\pi\right) = \mathbb{E}_{u^1 \sim \pi^1, \ldots, u^N \sim \pi^N, s \sim T} \left[\sum_{t=0}^{\infty} \gamma_t r_t \left(s_t, u_t^1, \ldots, u_t^N \right) \right] \tag{1}$$

Centralized Training and Decentralized Execution (CTDE). CTDE is a popular paradigm [32] which allows for the learning process to utilize additional state information [10]. Agents are trained in a centralized way, i.e., learning algorithms, to access all local action observation histograms, global states, and sharing gradients and parameters. In the execution stage, each individual agent can only access its local action observation history τ^i.

---

[1] These algorithms are based on the mixing network from QMIX, so we call the variants of QMIX.

QMIX and Monotonicity Constraint. To resolve the credit assignment problem in multi-agent learning, QMIX [20] learns a joint action-value function Q_{tot} which can be represented in Eq. 2:

$$
\begin{aligned}
Q_{tot}(s, \boldsymbol{u}; \boldsymbol{\theta}, \phi) =& g_\phi\left(s, Q_1\left(\tau^1, u^1; \theta^1\right), \ldots, Q_N\left(\tau^N, u^N; \theta^N\right)\right) \\
& \frac{\partial Q_{tot}(s, \boldsymbol{u}; \boldsymbol{\theta}, \phi)}{\partial Q_i\left(\tau^i, u^i; \theta^i\right)} \geq 0, \quad \forall i \in \mathcal{N}
\end{aligned}
\tag{2}
$$

where ϕ is the trainable parameter of the monotonic mixing network, which is a mixing network with monotonicity constraint , and θ^i is the parameter of the agent network i. Benefiting from the monotonicity constraint in Eq. 2, maximizing joint Q_{tot} is precisely the equivalent of maximizing individual Q_i, resulting in and allowing for optimal individual action to maintain consistency with optimal joint action. QMIX learns by sampling a multitude of transitions from the replay buffer and minimizing the mean squared temporal-difference (TD) error loss:

$$
\mathcal{L}(\theta) = \frac{1}{2} \sum_{i=1}^{b} \left[(y_i - Q_{tot}(s, u; \theta, \phi))^2\right]
\tag{3}
$$

where the TD target value $y = r + \gamma \max_{u'} Q_{tot}\left(s', u'; \theta^-, \phi^-\right)$ and θ^-, ϕ^- are the target network parameters copied periodically from the current network and kept constant for a number of iterations. However, the monotonicity constraint limits the mixing network's expressiveness, which may fail to

12	-12	-12
-12	0	0
-12	0	0

(a) Payoff matrix

-12	-12	-12
-12	0	**0**
-12	0	0

(b) QMIX: Q_{tot}

Table 1: A non-monotonic matrix game. Bold text indicates the reward of the argmax action.

learn in non-monotonic cases [12] [21]. Table 1a shows a non-monotonic matrix game that violates the monotonicity constraint. This game requires both robots to select the first action 0 (actions are indexed from top to bottom, left to right) in order to catch the reward 12; if only one robot selects action 0, the reward is -12. QMIX may learn an incorrect Q_{tot} which has an incorrect argmax action as shown in Table 1b.

3 RELATED WORKS

In this section, we describe the variants of QMIX which relaxing the monotonicity constraint. We explain the details of these algorithms and show the code resources in Appendix D.

Value-based Methods To enhance the expressive power of QMIX, Qatten [37] introduces an attention mechanism to enhance the expression of QMIX; QPLEX [32] transfers the monotonicity constraint from Q values to Advantage values [14]; QTRAN++ [25] and WQMIX [21] further relax the monotonicity constraint through a true value network and some theoretical constraints; however, Value-Decomposition Networks (VDNs) [29] only requires a linear decomposition where $Q_{tot} = \sum_i^N Q_i$, which can be seen as strengthening the monotonicity constraint.

Policy-based Methods LICA [39] completely removes the monotonicity constraint through a policy mixing critic. For other MARL policy-based methods, VMIX [27] combines the Advantage Actor-Critic (A2C) [26] with QMIX to extend the monotonicity constraint to value networks, i.e., replacing the value network (not Q value network) with the monotonic mixing network. DOP [33] learns the policy networks using the Counterfactual Multi-Agent Policy Gradients (COMA) [6] with the Q_i decomposed by QMIX. At last, we briefly describe the properties of these algorithms in Table 2.

All these algorithms show that their performance exceeds QMIX in SMAC, yet we find that they do not consider the impact of various code-level optimizations (Appendix A) in the implementations.

Algorithms	Type	Attention	Monotonic Constraint	Off-policy
VDNs	Value-based	No	Very Strong	Yes
QMIX	Value-based	No	Strong	Yes
Qatten	Value-based	Yes	Strong	Yes
QPLEX	Value-based	Yes	Medium	Yes
WQMIX	Value-based	No	Weak	Yes
VMIX	Policy-based	No	Strong	No
LICA	Policy-based	No	No	No
RIIT	Policy-based	No	Strong	Yes

Table 2: Properties of coopertive MARL algorithms.

Moreover, the performance of these algorithms is not even consistent in these papers. For example, in papers [32] and [33], QPLEX and DOP outperform QMIX, while in paper [17], both QPLEX and DOP underperform QMIX.

4 EXPERIMENTS SETUP

To facilitate the study of cooperation in complex multi-robots scenarios,, we simulate the interaction among the robots through two hard computer games.

4.1 BENCHMARK ENVIRONMENT

StarCraft Multi-Agent Challenge (SMAC) is used as our main benchmark testing environment, which is a ubiquitously-used multi-agent cooperative control environment for MARL algorithms [32; 20; 25; 21]. SMAC consists of a set of StarCraft II micro battle scenarios, whose goals are for allied robots to defeat enemy robots, and it classifies micro scenarios into *Easy*, *Hard*, and *Super Hard* levels. The simplest VDNs [29] can effectively solve the Easy scenarios. It is worth noting that QMIX and VDNs achieves a 0% win rate in three Super Hard scenarios $corridor$, $3s5z\_vs\_3s5z$, and $6h\_vs\_8z$ [22]. Therefore, we mainly investigate the Hard and Super Hard scenarios in SMAC.

Difficulty-Enhanced Predator-Prey (DEPP) In vanilla Predator-Prey [11], three cooperating agents control three robot predators to chase a faster robot prey (the prey acts randomly). The goal is to capture the prey with the fewest steps possible. We leverage two difficulty-enhanced Predator-Prey variants to test the algorithms: (1) the first variant of Predator-Prey (PP) [2] requires two predators to catch the prey at the same time to get a reward; (2) In the Continuous Predator-Prey (CPP) [17], the prey's policy is replaced by a hard-coded heuristic policy, i.e., at any time step, moving the prey to the sampled position with the largest distance to the closest predator,

4.2 EVALUATION METRIC

Our primary evaluation metric is the function that maps the steps for the environment observed throughout the training to the median winning percentage (episode return for Predator-Prey) of the evaluation. Just as in QMIX [20], we repeat each experiment with several independent training runs (five independent random experiments). To accurately evaluate the convergence performance of each algorithm, **eight rollout processes for parallel sampling** are used to obtain as many samples as possible from the environments at a high rate. Specifically, our experiments can collect 10 million samples within 9 hours with a Core i7-7820X CPU and a GTX 1080 Ti GPU.

5 EXPERIMENTS

Our experiments consist of two parts. The first part demonstrates the performance of several isolated tricks from the variants. The second part is the reconceptualization of the monotonicity constraint.

5.1 RETHINKING THE CODE-LEVEL OPTIMIZATIONS

The code-level optimizations are the tricks unaccounted for in the experimental design, but that might hold significant effects on the result. To better understand their influences on performance, we

perform ablation experiments on these tricks incrementally and provide some suggestions for tuning. We study the major optimizations here, and introduce the other tricks in Appendix A.

5.1.1 OPTIMIZER

Study description. QMIX and the majority of its variant algorithms use RMSProp to optimize neural networks as they prove stable in SMAC. We attempt to use Adam to optimize QMIX's neural network with quickly convergence benefiting from momentum:
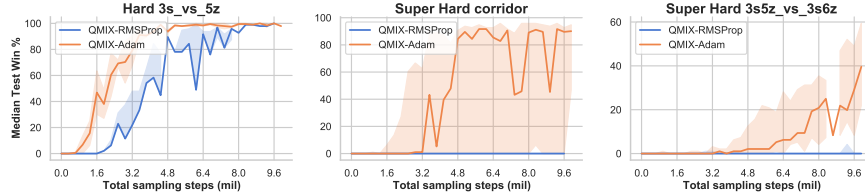

Figure 1: Adam significantly improves performance when samples are updated quickly.

Interpretation. Figure 1 shows that Adam [8] increases the win rate by 100% on the Super Hard map *corridor*. Adam boosts the network's convergence allowing for full utilization of the large quantity of samples sampled in parallel. We find that the Adam optimizer solves the problem posed by [27] in which QMIX does not work well under parallel training.

Recommendation. Use Adam with parallel training.

5.1.2 ELIGIBILITY TRACES

Study description. Eligibility traces such as TD(λ) [30], Peng's Q(λ) [18], and TB(λ) [19] achieve a balance between return-based algorithms (where return refers to the sum of discounted rewards $\sum_t \gamma^t r_t$) and bootstrap algorithms (where return refers to $r_t + V(s_{t+1})$), speeding up the convergence of reinforcement learning algorithms. Therefore, we study the application of Peng's Q(λ) for QMIX,
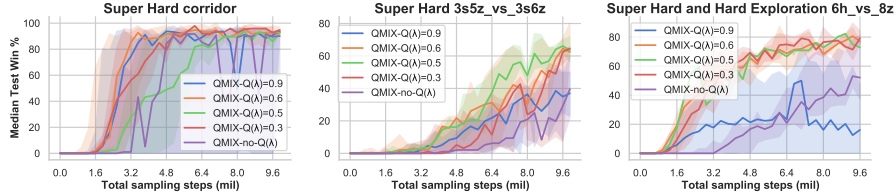

Figure 2: Experiments for Q(λ).

Figure 3: Q(λ) significantly improves performance of QMIX, but large values of λ lead to instability in the algorithm.

Interpretation. Q networks without sufficient training usually have a large bias that impacts bootstrap returns. Figure 3 shows that Q(λ) allows for faster convergence in our experiments by reducing this bias. However, large values of λ may lead to failed convergence due to the large variance. Figure 3 shows that when λ is set to 0.9, it has a detrimental impact on the performance of QMIX.

Recommendation. Use Q(λ) with a small value of λ.

5.1.3 REPLAY BUFFER SIZE

Study description. In single-agent Deep Q-networks (DQN), the replay buffer size is usually set to a large value. However, in multi-agent tasks, as the action space becomes larger than that of single-agent tasks, the distribution of samples changes more quickly. In this section, we study the impact of the replay buffer size on performance.

Interpretation. Figure 4 shows that a large replay buffer size causes instability in QMIX's learning. The causes of this phenomenon are as follows: (1) In multi-agent tasks, samples become obsolete
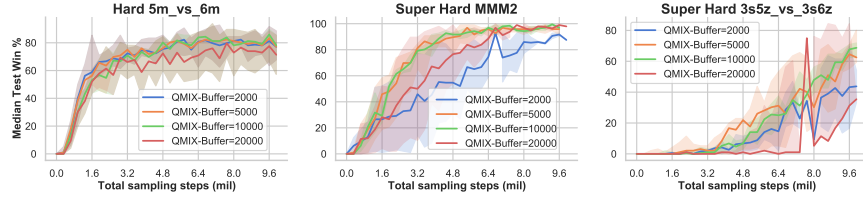
Figure 4: Setting the replay buffer size to 5000 episodes allows for QMIX's learning to be more stable than by setting it to 20000 episodes.

more quickly than in single-agent tasks. (2) Echoing in Sec. 5.1.1, Adam performs better with samples with fast updates. (3) When the sampling policy is far from the current policy, the return-based methods require importance sampling ratios, which is difficult to calculate in multi-agent learning.

Recommendation. Use a small replay buffer size.

5.1.4 ROLLOUT PROCESS NUMBER

Study description. When we collect samples in parallel as is done in A2C [26], it shows that when there is a defined total number of samples and an unspecified number of rollout processes, the median test performance becomes inconsistent. This study aims to perform analysis and provide insight on the impact of the number of processes on the final performance.
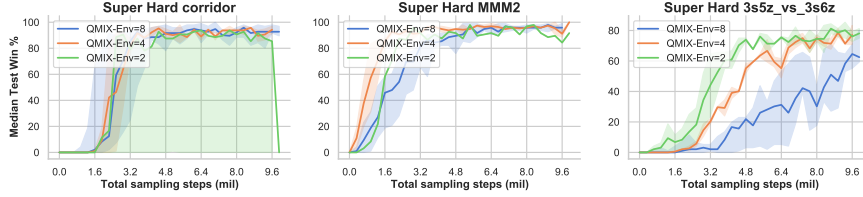

Figure 5: Given the total number of samples, fewer processes achieve better performance. We set the replay buffer size to be proportional to the number of processes to ensure that the novelty of the samples is consistent.

Interpretation. Under the A2C [14] training paradigm, the total number of samples can be calculated as $S = E \cdot P \cdot I$, where S is the total number of samples, E is the number of samples in each episode, P is the number of rollout processes, and I is the number of policy iterations. Figure 5 shows that we are given both S and E; the fewer the number of rollout processes, the greater the number of policy iterations [30]; a higher number of policy iterations leads to an increase in performance. However, it also causes both longer training time and decreased stability.

Recommendation. Use fewer rollout processes when samples are difficult to obtain, especially for real-world robot learning; otherwise, use more rollout processes.

5.1.5 HIDDEN SIZE

Study description. In deep reinforcement learning, researchers typically use smaller networks to train models, but the size of the neural network can also have an impact on algorithm performance. In this study, we analyze the network width of each layer of QMIX.

Interpretation. As shown in Figure 6, increasing the hidden size of the neural network of QMIX from 64 to 256 allows for a 18% increase in win rate in the hard scenario $3s5z\_vs\_3s6z$. More specifically, increasing the hidden size of RNNs is more helpful to improve the performance of QMIX than increasing the hidden size of mixing networks.

Recommendation. Increase the hidden size of QMIX to an appropriate value.
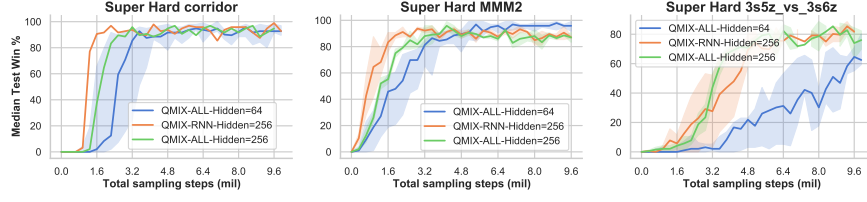
Figure 6: On the hard scenario $3s5z\_vs\_3s6z$, increasing the width of neural network significantly improves the performance of QMIX.

5.1.6 EXPLORATION STEPS

Study description. Some scenarios in SMAC are hard to explore, such as $6h\_vs\_8z$, so the settings of ϵ-greedy become critically important. In this study, we analyze the effect of ϵ anneal period on performance.
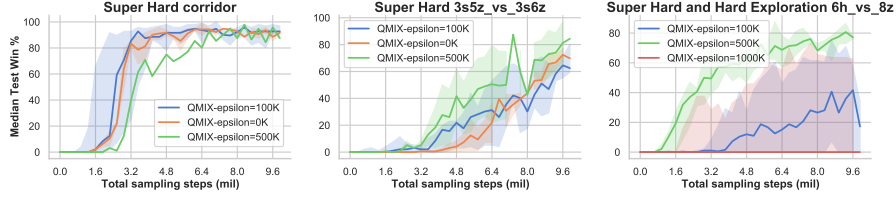

Figure 7: On the hard-to-explore scenario $6h\_vs\_8z$, defining a proper length for ϵ anneal period significantly improves performance.

Interpretation. As shown in Figure 7, increasing the length of the ϵ anneal period from 100K steps to 500K steps allows for a increase in win rate in the Super Hard scenario $6h\_vs\_8z$ (38%) and $3s5z\_vs\_3s6z$ (23%). However, increasing this value to 1000K instead causes the model to collapse.

Recommendation. Increase the value of the ϵ anneal period to an appropriate length on hard-to-explore scenarios.

5.1.7 OVERALL IMPACTS

Senarios	Difficulty	QMIX	Finetuned-QMIX
2s_vs_1sc	Easy	**100%**	**100%**
2s3z	Easy	**100%**	**100%**
1c3s5z	Easy	**100%**	**100%**
3s5z	Easy	**100%**	**100%**
10m_vs_11m	Easy	98%	**100%**
8m_vs_9m	Hard	84%	**100%**
5m_vs_6m	Hard	84%	**90%**
3s_vs_5z	Hard	96%	**100%**
bane_vs_bane	Hard	**100%**	**100%**
2c_vs_64zg	Hard	**100%**	**100%**
corridor	Super Hard	0%	**100%**
MMM2	Super Hard	98%	**100%**
3s5z_vs_3s6z	Super Hard	3%	**93%** (hidden_size = 256)
27m_vs_30m	Super Hard	56%	**100%**
6h_vs_8z	Super Hard	0%	**93%** ($\lambda = 0.3$)

Table 3: Best median test win rate of Finetuned-QMIX and QMIX (batch size=128) in all scenarios.

Then we finetuned all these hyperparameters (besides the number of rollout processes) of QMIX for each scenarios of SMAC. As shown in Table 3, Finetuned-QMIX attains higher win rates in all hard and super hard SMAC scenarios, far exceeding vanilla QMIX.

5.2 RETHINKING THE MONOTONICITY CONSTRAINT

In this subsection, as the past studies evaluate the performance of QMIX's variants with inconsistent implementation tricks, we retested their performance based on the normalized tricks (details in Appendix B). In addition, RIIT and VMIX are demonstrated to further study the effects of the monotonicity constraint.

5.2.1 RE-EVALUATION

Scenarios	Difficulty	Value-based					Policy-based			
		QMIX	VDNs	Qatten	QPLEX	WQMIX	LICA	VMIX	DOP	RIIT
2c_vs_64zg	Hard	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	98%	84%	**100%**
8m_vs_9m	Hard	**100%**	**100%**	**100%**	95%	95%	48%	75%	96%	95%
3s_vs_5z	Hard	**100%**	**100%**	**100 %**	**100%**	**100%**	3%	96%	**100%**	96%
5m_vs_6m	Hard	**90%**	**90%**	**90%**	**90%**	**90%**	53%	9%	63%	67%
3s5z_vs_3s6z	S-Hard	**75%**	43%	62%	68%	56%	0%	56%	0%	**75%**
corridor	S-Hard	**100%**	98%	**100%**	96%	96%	0%	0%	0%	**100%**
6h_vs_8z	S-Hard	84%	**87%**	82%	78%	75%	4%	80%	0%	19%
MMM2	S-Hard	**100%**	96%	**100%**	**100%**	96%	0%	70%	3%	**100%**
27m_vs_30m	S-Hard	**100%**	**100%**	**100%**	**100%**	**100%**	9%	93%	0%	93%
Discrete PP	-	**40**	39	-	39	39	30	39	38	38
Avg. Score	(Hard+)	**94.9%**	91.2%	92.7%	92.5%	90.5%	29.2%	67.4%	44.1%	84.0%

Table 4: Median test winning rate (episode return) of MARL algorithms with normalized tricks. S-Hard denotes Super Hard. We compare their performance in the most difficult scenarios of SMAC and the Discrete PP.

We then normalize the tricks for all these algorithms for the re-evaluation, i.e, we perform grid search schemes on a typical hard environment (5m_vs_6m) and super hard environment (3s5z_vs_3s6z) to find **a general set of** hyperparameters for each algorithm (details in Appendix B.1). As shown in Table 4, the test results on the hardest scenarios in SMAC and DEPP demonstrate that, (1) The performance of values-based methods and VMIX with normalized tricks exceeds the test results in the past literatures [22; 32; 17; 21; 28] (details in Appendix B.2). (2) **QMIX outpeforms all its variants**. (3) The linear VDNs is also relatively effective. (4) The performance of the algorithm becomes progressively worse as the monotonicity constraint decreases (QMIX > QPLEX > WQMIX > LICA, details in Appendix D.8) in the benchmark environment.

The experimental results, specifically (2), (3) and (4), show that these variants of QMIX that relax the monotonicity constraint do not obtain better performance than QMIX in some purely cooperative tasks, either SMAC or DEPP.

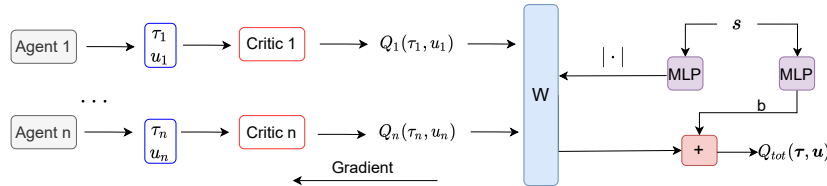### 5.2.2 ALBATION STUDIES OF MONOTONICITY CONSTRAINT

Figure 8: Architecture for RIIT: $|\cdot|$ **denotes absolute value operation**, implementing the monotonicity constraint of QMIX. W denotes the non-negative mixing weights. Agent i denotes the policy network which can be trained end-to-end by maximizing the Q_{tot}.

We further study the impact of monotonicity constraint tasks via comparing the performance of adding or removing the constraint. An end-to-end Actor-Critic method, RIIT, is proposed. Specifically, we use the monotonic mixing network as a critic network, shown in Figure 8. Then, in Eq. 4, with a trained critic $Q_{\theta_c}^\pi$ estimate, the decentralized policy networks $\pi_{\theta_i}^i$ can then be optimized end-to-end simultaneously by maximizing $Q_{\theta_c}^\pi$ with the policies $\pi_{\theta_i}^i$ as inputs. Since RIIT is trained end-to-end,
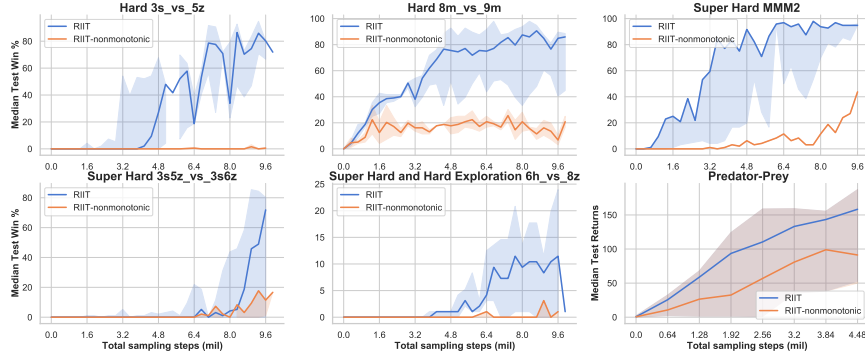
Figure 9: Comparing RIIT w./ and w./o. monotonicity constraint (remove absolute value operation) on SMAC and Continuous Predator-Prey.

it may also be used for continuous control tasks. It is worth stating that the item $\mathbb{E}_i\left[\mathcal{H}\left(\pi^i_{\theta_i}\left(\cdot\mid z^i_t\right)\right)\right]$ is the Adaptive Entropy [39], and we use a two-stage approach to train the actor-critic network, described in detail in Appendix C.

$$\max_\theta \mathbb{E}_{t,s_t,\tau^1_t,\ldots,\tau^n_t}\left[Q^\pi_{\theta_c}\left(s_t,\pi^1_{\theta_1}\left(\cdot\mid\tau^1_t\right),\ldots,\pi^n_{\theta_n}\left(\cdot\mid\tau^n_t\right)\right)+\mathbb{E}_i\left[\mathcal{H}\left(\pi^i_{\theta_i}\left(\cdot\mid\tau^i_t\right)\right)\right]\right] \tag{4}$$

The monotonicity constraint on the critic (Figure 8) is theoretically no longer required as the critic is not used for greedy action selection. We can evaluate the effects of the monotonicity constraint by removing the absolute value operation in the monotonic mixing network. In this way, RIIT can also be easily extended to non-monotonic tasks. Figure 9 demonstrates that the monotonicity constraint significantly improves the performance of RIIT. Table 4 also presents that RIIT performs best among these policy-based algorithms.
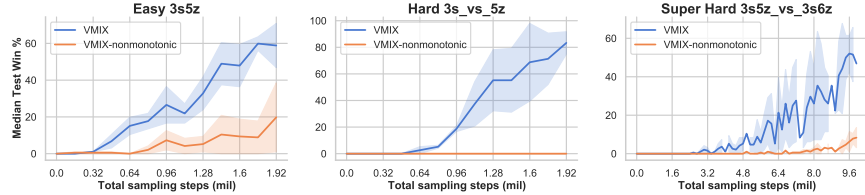

Figure 10: Comparing VMIX with and without monotonicity constraint on SMAC.

To explore the generality of monotonicity constraints, we extend the above experiments to VMIX [28]. VMIX adds the monotonicity constraint to the value network (not Q value networks) of A2C. (details in Appendix D.7) VMIX learns the policy of each agent by advantage-based policy gradient [14]; therefore, the monotonicity constraint is not necessary for greedy action selection either. We can evaluate the effects of the monotonicity constraint by removing the absolute value operation in Figure 12. The result from Figure 10 shows that the monotonicity constraint improves the sample efficiency in value networks. The above experimental results indicate that the monotonicity constraint can improve the sample efficiency in some multi-robot cooperative tasks, such as SMAC and DEPP.

6 DISCUSSION

To better understand the monotonicity constraint, we discuss the following two questions with theoretical analysis. **Ques.1** Why can SMAC be represented well by monotonic mixing networks? **Ques.2** Why can the monotonicity constraint improve the sample efficiency in SMAC? To coherently answer the above questions, we give the following definitions and propositions. It is worth noting that the core assumption is that the joint action-value function Q_{tot} can be represented by a non-linear mapping $f_\phi(s;Q_1,Q_2,...Q_N)$, *but without the monotonicity constraint.*

Definition 1. *Cooperative tasks. For a task with N agents ($N > 1$), all agents have a common goal.*

Definition 2. *Semi-cooperative Tasks. Given a cooperative task with a set of agents $\mathbb{N}$. For all states s of the task, if there is a subset $\mathbb{K} \subseteq \mathbb{N}$, $\mathbb{K} \neq \varnothing$, where the $Q_i, i \in \mathbb{K}$ increases while the other $Q_j, j \notin \mathbb{K}$ are fixed, this will lead to an increase in Q_{tot}.*

As a counterexample, the collective action problem (social dilemma) is not Semi-cooperative task. i.e., since the Q value may not include future rewards when $\gamma < 1$, the collective interest in the present may be detrimental to the future interest.

Definition 3. *Competitive Cases. Given two agents i and j, we say that agents i and j are competitive if either an increase in Q_i leads to a decrease in Q_j or an increase in Q_j leads to a decrease in Q_i.*

Definition 4. *Purely Cooperative Tasks. Semi-cooperative tasks without competitive cases.*

As an counterexample, the matrix game as in Table 1a is not a purely cooperative task. Because of the random action sampling in reinforcement learning, we cannot guarantee that the agents share the same preferences. If one agent prefers action 0 (Like hunting) and the other agent prefers action 1 or 2 (Like sleeping or entertaining), they will have a conflict of interest (Those who like to sleep will cause the hunter to fail to catch the prey).

Proposition 1. *Purely Cooperative Tasks can be represented by monotonic mixing networks.*

Proof. Since the QMIX's mixing network is a universal function approximator of monotonic functions, for a Semi cooperative task, if there is a case (state s) that cannot be represented by a monotonic mixing network, i.e., $\frac{\partial Q_{tot}(s)}{\partial Q_i} < 0$, then an increase in Q_i must lead to a decrease in $Q_j, j \neq i$ (since there is no Q_j decrease, by Def. 2, the constraint $\frac{\partial Q_{tot}(s)}{\partial Q_i} < 0$ does not hold). Therefore, by Def. 3 this cooperative task has a competitive case which means it is not a purely cooperative task. □

For answering Ques.1: According to the Proposition 1, we need to explain why SMAC can be seen as a purely cooperative task environment. SMAC mainly uses a shaped reward signal calculated from the hit-point damage dealt, some positive reward after having enemy units killed and a positive bonus for winning the battle. In practice, we can decompose the hit-point damage dealt linearly, and divide the units killed rewards to the agents near the enemy evenly, the victory rewards to all agents. **This fair reward decomposition can be interpreted as purely cooperative.** Futuremore, as $Q_\pi(s, u) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s, u]$, the reward is linearly assignable meaning that Q value is linearly assignable, which also explains why the VDNs also work well in SMAC (Table. 4).

For answering Ques.2: Just as in RIIT's implementation (Figure 8) where the monotonicity constraint reduces the range of values of each mixing weight by half, the hypothesis space is assumed to decrease exponentially by $(\frac{1}{2})^N$ (N denotes the number of weights). Note that the Q value decomposition mapping of the SMAC is a subset of the hypothesis space of QMIX's mixing network. Therefore, using the monotonicity constraint can allow for avoiding searching invalid parameters, leading to a significant improvement in sampling efficiency.

Our analysis shows that QMIX works well if a multi-agent task can be interpreted as purely cooperative, even if it can also be interpreted as competitive. **That is, QMIX will try to find a purely cooperative interpretation for a complex multi-agent task.**

7 CONCLUSION

In this paper, we investigate the influence of certain code-level optimizations on the performance of QMIX and provide tuning optimizations suggestions. Then, we find that monotonicity constraint can improve sample efficiency in SMAC and DEPP, benefiting to the real-world robot learning. Our analysis imply that we can design reward functions in the real multi-agent task that can be interpreted as purely cooperative, improving the learning sample efficiency of the MARL. Meanwhile, we also believe that the variants that relax monotonicity constraint of QMIX might be well-suited for the mutil-agent tasks which cannot be interpreted as purely cooperative. In addition, we are hopeful that this paper will call on the community to be more fair in comparing the performance of algorithms.

REFERENCES

[1] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study. *arXiv:2006.05990*, 2020.

[2] Wendelin Boehmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In *ICML 2020, 13-18 July 2020, Virtual Event*, pp. 980–991, 2020.

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

[4] Karl Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. *arXiv preprint arXiv:2009.04416*, 2020.

[5] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO. *arXiv:2005.12729*, 2020.

[6] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI-18, New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 2974–2982. AAAI Press, 2018.

[7] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Guided deep reinforcement learning for swarm systems. *arXiv preprint arXiv:1709.06011*, 2017.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR 2015, San Diego, CA, USA, May 7-9, 2015*, 2015.

[9] Tadashi Kozuno, Yunhao Tang, Mark Rowland, Rémi Munos, Steven Kapturowski, Will Dabney, Michal Valko, and David Abel. Revisiting peng's q (λ) for modern reinforcement learning. *arXiv preprint arXiv:2103.00107*, 2021.

[10] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016. ISSN 09252312.

[11] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NeurIPS 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 6379–6390, 2017.

[12] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. MAVEN: multi-agent variational exploration. In *NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 7611–7622, 2019.

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

[14] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1928–1937, 2016.

[15] Ofir Nachum, Michael Ahn, Hugo Ponte, Shixiang Gu, and Vikash Kumar. Multi-agent manipulation via locomotion using hierarchical sim2real. *arXiv preprint arXiv:1908.05224*, 2019.

[16] Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Pomdps for robotic tasks with mixed observability. 5:4, 2009.

[17] Bei Peng, Tabish Rashid, Christian A Schroeder de Witt, Pierre-Alexandre Kamienny, Philip HS Torr, Wendelin Böhmer, and Shimon Whiteson. Facmac: Factored multi-agent centralised policy gradients. *arXiv e-prints*, pp. arXiv–2003, 2020.

[18] Jing Peng and Ronald J Williams. Incremental multi-step q-learning. In *Machine Learning Proceedings 1994*, pp. 226–232. Elsevier, 1994.

[19] Doina Precup, Richard S. Sutton, and Satinder P. Singh. Eligibility traces for off-policy policy evaluation. In *(ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pp. 759–766. Morgan Kaufmann, 2000.

[20] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In *ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 4292–4301, 2018.

[21] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted QMIX: Expanding Monotonic Value Function Factorisation. *arXiv preprint arXiv:2006.10800*, 2020.

[22] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *arXiv preprint arXiv:1902.04043*, 2019.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[24] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Hostallero, and Yung Yi. QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 5887–5896, 2019.

[25] Kyunghwan Son, Sungsoo Ahn, Roben Delos Reyes, Jinwoo Shin, and Yung Yi. QTRAN++: Improved Value Transformation for Cooperative Multi-Agent Reinforcement Learning. *arXiv:2006.12010*, 2020.

[26] Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*, 2018.

[27] Jianyu Su, Stephen Adams, and Peter A Beling. Value-decomposition multi-agent actor-critics. *arXiv preprint arXiv:2007.12306*, 2020.

[28] Jianyu Su, Stephen Adams, and Peter A. Beling. Value-Decomposition Multi-Agent Actor-Critics. *arXiv:2007.12306*, 2020.

[29] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-Decomposition Networks For Cooperative Multi-Agent Learning. *arXiv preprint arXiv:1706.05296*, 2017.

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

[31] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pp. 330–337, 1993.

[32] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. QPLEX: Duplex Dueling Multi-Agent Q-Learning. *arXiv:2008.01062*, 2020.

[33] Yihan Wang, Beining Han, Tonghan Wang, Heng Dong, and Chongjie Zhang. Off-Policy Multi-Agent Decomposed Policy Gradients. *arXiv:2007.12322*, 2020.

[34] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1995–2003.

[35] Ermo Wei, Drew Wicke, David Freelan, and Sean Luke. Multiagent Soft Q-Learning. *arXiv preprint arXiv:1804.09817*, 2018.

[36] Yuchen Xiao, Joshua Hoffman, and Christopher Amato. Macro-action-based deep multi-agent reinforcement learning. In *Conference on Robot Learning*, pp. 1146–1161. PMLR, 2020.

[37] Yaodong Yang, Jianye Hao, Ben Liao, Kun Shao, Guangyong Chen, Wulong Liu, and Hongyao Tang. Qatten: A General Framework for Cooperative Multiagent Reinforcement Learning. *arXiv preprint arXiv:2002.03939*, 2020.

[38] Chongjie Zhang and Victor R. Lesser. Coordinated multi-agent reinforcement learning in networked distributed pomdps. In *AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.

[39] Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. Learning Implicit Credit Assignment for Multi-Agent Actor-Critic. *arXiv preprint arXiv:2007.02529*, 2020.

[40] Ming Zhou, Jun Luo, and Julian Villella et al. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving, 2020.

A   RETHINKING THE CODE-LEVEL OPTIMIZATIONS (EXTENSION OF SEC. 5.1)

Engstrom *et.al* [5] investigates code-level optimizations based on PPO [23] implementation, and concludes that the majority of performance differences between PPO and TRPO originate from code-level optimizations. Andrychowicz *et. al* [1] investigates the influence of code-level optimizations on the performance of PPO and provides tuning optimizations. These optimizations include: (1) Adam and Learning rate annealing. (2) Orthogonal initialization and Layer scaling. (3) Observation normalization. (4) Value normalization. (5) N-step returns (eligibility traces). (6) Reward scaling. (7) Reward clipping. (8) Neural Network Size. Using a subset of the whole code-level optimizations, specifically shown in Sec. 5.1, we enabled QMIX to solve almost all scenarios of SMAC.

We also propose a simple trick, i.e, rewards shaping, to help QMIX learning in a non-monotonic environment.

A.1   REWARDS SHAPING

Study description. Table 1a shows a non-monotonic case that QMIX cannot solve. However, the reward function in MARL is defined by the user; we investigate whether QMIX can learn a correct argmax action by reshaping the task's reward function without changing its goal.

12.0	-0.5	-0.5
-0.5	0	0
-0.5	0	0

(a) Reshaped Payoff matrix

12.0	-0.3	-0.3
-0.3	-0.3	-0.3
-0.3	-0.3	-0.3

(b) QMIX: Q_{tot}

Table 5: A non-monotonic matrix game in which we reshape the reward by replacing the insignificant reward -12 (in Table 1a) with reward -0.5. QMIX learns a Q_{tot} which has a correct argmax. Bold text indicates argmax action's reward.

Interpretation. The reward -12 in Table 1a does not assist the agents in finding the optimal solution; as shown in Table 5, this non-monotonic matrix may be solved by simply replacing the insignificant reward -12 with -0.5. The reward shaping may also help QMIX learn more effectively in other non-monotonic tasks.

Recommendation. Increase the scale of the important rewards of the tasks and reduce the scale of rewards that may cause disruption.

A.2   PENG'S Q(λ)

We briefly introduce Peng's Q(λ) here, TD(λ) can be expressed as Eq. 5:

$$
G_s^\lambda \doteq (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{s:s+n}
$$
$$
G_{s:s+n} \doteq \sum_{t=s}^{s+n} \gamma^{t-s} r_t + \gamma^{n+1} V\left(s_{s+n+1}, u\right)
\tag{5}
$$

Peng's Q(λ) replaces the V value of the next state with the max Q value, as shown in Eq. 6:

$$
G_{s:s+n} \doteq \sum_{t=s}^{s+n} \gamma^{t-s} r_t + \gamma^{n+1} \max_u Q\left(s_{s+n+1}, u\right)
\tag{6}
$$

where λ is the discount factor of the traces and $\left(\prod_{s=1}^{t} \lambda\right) = 1$ when $t = 0$. When λ is set to 0, it is equivalent to 1-step bootstrap returns. When λ is set to 1, it is equivalent to Monte Carlo [30] returns.

[9] show that while Peng's Q(λ) does not learn optimal policies under arbitrary behavior policies, a convergence guarantee can be recovered if the behavior policy tracks the target policy, as is often the case in practice.

B EXPERIMENTAL DETAILS

B.1 HYPERPARAMETERS

Tricks	Value-based (VB)	Policy-bassed (PG)
Optimizer	Adam, RMSProp	Adam, RMSProp
Learning Rates	0.0005, 0.001	0.0005, 0.001, (and 0.0001 for DOP)
Batch Size(episodes)	32, 64, 128	32, 64
Replay Buffer Size	5000, 10000, 20000	2000, 5000, 10000, 20000
Q(λ), TD(λ)	0, 0.3, 0.6, 0.9	0, 0.3, 0.6, 0.9
(Adaptive) Entropy	-	0.01, 0.03, 0.06
ϵ Anneal Steps	50K, 100K, 500K, 1000K	100K, 500K for DOP

Table 6: Hyperparameters Search on SMAC.

In this section, we present our tuning process. We get the optimal hyperparameters for each algorithm by the grid search, shown in Table 6. Specifically,

1. For experiments in Sec. 5.2.1, we perform grid search schemes on a typical hard environment (5m_vs_6m) and super hard environment (3s5z_vs_3s6z) to find a **general** set of hyperparameters for each algorithm. In this way, we can evaluate the robustness of these MARL algorithms.

2. For experiments in Sec. 5.1.7, we perform hyperparameter search on each scenarios for QMIX to demonstrate the best performance of QMIX.

Algorithms	LICA	**OurLICA**	DOP	**OurDOP**	RIIT
Optimizer	Adam	**Adam**	RMSProp	RMSProp	Adam
Batch Size(episodes)	32	32	Off=32, On=16	**Off=64, On=32**	Off=64, On=32
TD(λ)	0.8	**0.6**	0.8, TB(λ=0.93)	0.8, TB(λ=0.93)	0.6
Adaptive Entropy	0.06	0.06	-	-	0.03
ϵ Anneal Steps	-	-	500K (double)	500K (double)	-
Critic-Net Size	29696K	**389K**	122K	122K	69K
Rollout Processes	32	**8**	4	**8**	8

(a) Setting of Policy-based algorithms.
double: DOP first adds noise to the output of the policy network, then mask invalid actions and adds noise to the probabilities again.

Algorithms	QMIX	**OurQMIX**	Qatten	**OurQatten**	QPLEX	**OurQPLEX**
Optimizer	RMSProp	**Adam**	RMSProp	**Adam**	RMSProp	**Adam**
Batch Size (epi.)	128	**128**	32	**128**	32	**128**
Q(λ)	0	**0.6**	0	**0.6**	0	**0.6**
Attention Heads	-	-	4	4	10	**4**
Mixing-Net Size	41K	41K	58K	58K	476K	**152K**
ϵ Anneal Steps	50K $\rightarrow$ **500K for** $6h\_vs\_8z$**, 100 K for others**					
Rollout Processes	8	**8**	1	**8**	1	**8**

(b) Setting of Value-based algorithm.

Table 7: Hyperparameters Settings.

Table 7a and 7b shows our general settings for the these algorithms. The network size is calculated under $6h\_vs\_8z$, where adding *Our* denotes the new hyperparameter settings. Next, we describe in detail the setting of these hyperparameters,

Neural Network Size We first ensure the network size is the same order of magnitude, which means that we decrease the critic-net size of LICA from 29696K to 389K, and we use 4 attention heads

leading the mixing-net size of QPLEX from 476K to 152K. All the agent networks are the same as those found in QMIX [20].

Optimizer & Learning Rate We use Adam to optimize all networks, except VMIX (works better with RMSProp), as it may accelerate the convergence of the algorithms. Furthermore, we use different learning rates for each algorithm: (1) For all value-based algorithms, neural networks are trained with 0.001 learning rate. (2) For LICA, we set the learning rate of the agent network to 0.0025 and the critic network's learning rate to 0.0005. (3) For RIIT and VMIX, we set the learning rates to 0.001.

Batch Size We find that a large batch size helps to improve the stability of the algorithms. Therefore, for value-based algorithms, we set the batch size to 128. For the policy-based algorithms, we set the batch size to 64/32 (Offline/Online training) due to the fact that online update requires only the newest data.

Replay Buffer Size As discussed in Appendix. 5.1.3, a small replay buffer size facilitates the convergence of the MARL algorithms. Therefore, for SMAC, the size of all replay buffers is set to 5000 episodes. For Predator-Prey, we set the buffer size to 1000 episodes.

Exploration As discussed in Appendix. 5.1.6, we use ϵ-greedy action selection, decreasing ϵ from 1 to 0.05 over n-time steps (n can be found in Table 7b) for value-based algorithms. We use the Adaptive Entropy [39] (Appendix. D.6) for all policy-based algorithms, except VMIX (works better with ordinary entropy and annealing noise), because it facilitates the automatic adjustment of the size of the entropy loss in different scenarios. Specialy, we add the Adaptive Entropy to DOP to prevent it from crashing in SMAC.

N-step returns We find that the λ values of $Q(\lambda)$ and $TD(\lambda)$ are hevily depend on the scenario. We are using $\lambda = 0.6$ for all tasks as the value works stably in most scenarios. However, for the on-policy method VMIX, we set $\lambda = 0.8$.

Rollout Processes Number For SMAC and Discrete PP, 8 rollout processes for parallel sampling are used to obtain as many samples as possible from the environments at a high rate. This also ensures that all the algorithms share the same number of policy iterations and sample size (10 million). For the non-monotonic matrix games, we set the processes number to 32. At last, 4 rollout processesare used for Continuous PP.

Other Settings We set all discount factors $\gamma = 0.99$. We update the target network every 200 episodes. We find that the optimal hyperparameters of the value-based algorithms are similar due to the fact that they share the same basic architecture and training paradigm. Therefore, the settings for VDNs and WQMIX are the same as for QMIX. Specifically, we use OW-QMIX, detailed in D.5, in WQMIX as the baseline.

Note that our experimental results are not directly comparable with the previous works (which use SC2.4.6), as we use StarCraft 2 (SC2.4.10) in the latest PyMARL.

B.2 THE PERFORMANCE OF ORIGINAL ALGORITHMS

In this section, we compare performance of the original algorithms with third-party experimental results, i.e. experimental results of the paper citing the algorithm.

For VDNs and QMIX, the original SMAC paper [22] shows that VDNs and QMIX do not perform well in hard and super hard scenarios. For Qatten, the experiments in [32] demonstrates that the performance of Qatten is worse than vanilla QMIX. [17] demonstrates that QPLEX and DOP does not work well in hard and super hard scenarios in SMAC, and the their performance is worse than vanilla QMIX. It is interesting that WQMIX [21] shows the poor performance of WQMIX in super hard scenarios $3s5z\_vs\_3s6z$ and $corridor$. The original test results in LICA are not considered as 64 million samples are used in their experiments.

However, after our hyperparameter tuning, all the value-based methods perform well in Hard and Super Hard scenarios. This shows that our hyperparameters does improve their performance.

C RIIT

In this section, we show the pseudo-code for the training procedure of RIIT. (1) Training the critic network with offline samples and 1-step TD error loss improves the sample efficiency for critic networks; (2) Training policy networks end-to-end and critic with TD(λ) and online samples improves learning stability of RIIT [2].

Algorithm 1 Optimization Procedure for RIIT

Initialize offline replay memory D and online replay memory D'.
Randomly initialize θ and ϕ for the policy networks and the mixing critic respectively.
Set $\phi^- \leftarrow \phi$.
while not terminated **do**
    # Off-policy stage
    Sample b episodes $\tau_1, ..., \tau_b$ with $\tau_i = \{s_{0,i}, o_{0,i}, u_{0,i}, r_{0,i}, ..., s_{T,i}, o_{T,i}, u_{T,i}, r_{T,i}\}$ from offline replay memory D.
    Update the monotonic mixing network with $y_{t,i}$ calculated by 1-step bootstrap return ($y_{t,i} = r_{t,i} + \gamma Q_{\phi^-}^{\pi}(s_{t+1}, \vec{u}_{t+1})$):

$$\nabla_\phi \frac{1}{bT} \sum_{i=1}^{b} \sum_{t=1}^{T} \left(y_{t,i} - Q_\phi^\pi \left(s_{t,i}, u_{t,i}^1, ..., u_{t,i}^n \right) \right)^2. \tag{7}$$

    # On-policy stage
    Sample b episodes $\tau_1, ..., \tau_b$ with $\tau_i = \{s_{0,i}, o_{0,i}, u_{0,i}, r_{0,i}, ..., s_{T,i}, o_{T,i}, u_{T,i}, r_{T,i}\}$ from online replay memory D'.
    Update the monotonic mixing network with $y_{t,i}^{TD(\lambda)}$ calculated by TD(λ):

$$\nabla_\phi \frac{1}{bT} \sum_{i=1}^{b} \sum_{t=1}^{T} \left(y_{t,i}^{TD(\lambda)} - Q_\phi^\pi \left(s_{t,i}, u_{t,i}^1, ..., u_{t,i}^n \right) \right)^2. \tag{8}$$

    Update the decentralized policy networks end-to-end by maximizing the Q value, with Adaptive Entropy Loss (the entropy normalized by its module length) :

$$\nabla_\theta \frac{1}{bT} \sum_{i=1}^{b} \sum_{t=1}^{T} \left(-Q_\phi^\pi \left(s_{t,i}, \pi_{\theta_1}^1(\cdot|z_{t,i}^1), ..., \pi_{\theta_n}^n(\cdot|z_{t,i}^n) \right) - \frac{1}{n} \sum_{a=1}^{n} \mathcal{H} \left(\pi_{\theta_a}^a(\cdot|z_{t,i}^a) \right) \right). \tag{9}$$

    if at target update interval **then**
        Update the target mixing network $\phi^- \leftarrow \phi$.
    end if
end while

D CTDE ALGORITHMS

D.1 IQL

Independent Q-learning (IQL) [31] breaks down a multi-agent task into a series of simultaneous single-agent tasks that share the same environment, just like multi-agent Deep Q-networks (DQN) [13]. DQN represents the action-value function with a deep neural network parameterized by θ. DQN uses a replay buffer to store transition tuple $\langle s, u, r, s' \rangle$, where state s' is observed after taking action u in state s and obtaining reward r. However, IQL does not address the non-stationarity introduced due to the changing policies of the learning agents. Thus, unlike single-agent DQN, there is no guarantee of convergence even at the limit of infinite exploration.

---

[2][4] shows that actor-networks generally have a lower tolerance for sample reuse than critic networks; and for RIIT, our empirical evidence shows that TD(λ) is not stable in the offline samples.

D.2 VDNs

By contrast, Value decomposition networks (VDNs) [3] [29] seek to learn a joint action-value function $Q_{tot}(\boldsymbol{\tau}, \mathbf{u})$, where $\boldsymbol{\tau} \in \mathbf{T} \equiv \mathcal{T}^n$ is a joint action- observation history and $\mathbf{u}$ is a joint action. It represents Q_{tot} as the sum of individual value functions $Q_a\left(\tau^i, u^i; \theta^i\right)$:

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \sum_{i=1}^{n} Q_i\left(\tau^i, u^i; \theta^i\right).$$

D.3 QATTEN

Qatten [4] [37], introduces an attention mechanism into the monotonic mixing network of QMIX:

$$Q_{tot} \approx c(s) + \sum_{h=1}^{H} w_h \sum_{i=1}^{N} \lambda_{i,h} Q^i \tag{10}$$

$$\lambda_{i,h} \propto \exp\left(e_i^T W_{k,h}^T W_{q,h} e_s\right) \tag{11}$$

where $w_h = \left|f^{NN}(s)\right|_h$, $W_{q,h}$ transforms e_s into a global query, and $W_{k,h}$ transforms e_i into an individual key. The e_s and e_i may be obtained by an embedding transformation layer for the true global state s and the individual state s_i.

D.4 QPLEX

QPLEX [5] [32] decomposes Q values into advantages and values based on Qatten, similar to Dueling-DQN [34]:

$$\text{(Joint Dueling) } Q_{tot}(\tau, u) = V_{tot}(\tau) + A_{tot}(\tau, u)$$
$$V_{tot}(\boldsymbol{\tau}) = \max_{u'} Q_{tot}\left(\boldsymbol{\tau}, \boldsymbol{u}'\right) \tag{12}$$

$$\text{(Individual Dueling) } Q_i\left(\tau_i, u_i\right) = V_i\left(\tau_i\right) + A_i\left(\tau_i, u_i\right)$$
$$V_i\left(\tau_i\right) = \max_{u'} Q_i\left(\tau_i, u_i'\right) \tag{13}$$

$$\frac{\partial A_{tot}(s, \boldsymbol{u}; \boldsymbol{\theta}, \phi)}{\partial A_i\left(\tau^i, u^i; \theta^i\right)} \geq 0, \quad \forall i \in \mathcal{N} \tag{14}$$

In other words, Eq. 14 (advantage-based monotonicity) transfers the monotonicity constraint from Q values to advantage values. QPLEX thereby reduces limitations on the mixing network's expressiveness.

D.5 WQMIX

WQMIX [6] [21], just like Optimistically-Weighted QMIX (OW-QMIX), uses different weights for each sample to calculate the squared TD error of QMIX:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} w(s, \mathbf{u})\left(Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s) - y_i\right)^2 \tag{15}$$

---

[3]VDN code: https://github.com/oxwhirl/pymarl
[4]Qatten code: https://github.com/simsimiSION/pymarl-algorithm-extension-via-starcraft
[5]QPLEX code: https://github.com/wjh720/QPLEX
[6]WQMIX code: https://github.com/oxwhirl/wqmix

$$w(s, \mathbf{u}) = \begin{cases} 1 & Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s) < y_i \\ \alpha & \text{otherwise.} \end{cases} \tag{16}$$

Where $\alpha \in (0, 1]$ is a hyperparameter and y_i is the true target Q value. WQMIX prefers those optimistic samples (true returns are larger than predicted), i.e., decreasing the weights of samples with non-optimistic returns. More critically, WQMIX uses an unconstrained true Q Network as a target network to guide the learning of QMIX. The authors prove that this approach can resolve the estimation errors of QMIX in the non-monotonic case.

D.6 LICA

LICA [7] [39] completely removes the monotonicity constraint through a policy mixing critic, as shown in Figure 11:
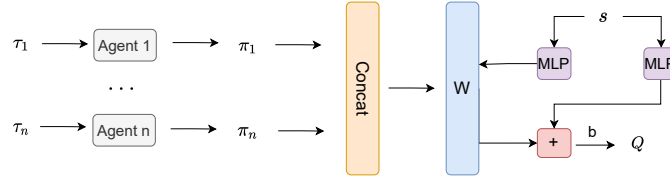

Figure 11: Architecture for LICA. LICA's mixing critic maps policy distribution to the Q value directly, in effect obviating the monotonicity constraint.

LICA's mixing critic is trained using squared TD error. With a trained critic estimate, decentralized policy networks may then be optimized end-to-end simultaneously by maximizing $Q_{\theta_c}^{\pi}$ with the stochastic policies $\pi_{\theta_i}^i$ as inputs:

$$\max_{\theta} \mathbb{E}_{t, s_t, u_t^1, \dots, \tau_t^n} [Q_{\theta_c}^{\pi} \left(s_t, \pi_{\theta_1}^1 \left(\cdot \mid \tau_t^1\right), \dots, \pi_{\theta_n}^n \left(\cdot \mid \tau_t^n\right)\right) - \mathbb{E}_i \left[\mathcal{H} \left(\pi_{\theta_i}^i \left(\cdot \mid \tau_t^i\right)\right)\right]] \tag{17}$$

where the gradient of entropy item $\mathbb{E}_i \left[\mathcal{H} \left(\pi_{\theta_i}^i \left(\cdot \mid z_t^i\right)\right)\right]$ is normalized by taking the quotient of its own modulus length: Adaptive Entropy (Adapt Ent). Adaptive Entropy automatically adjusts the coefficient of entropy loss in different scenarios.
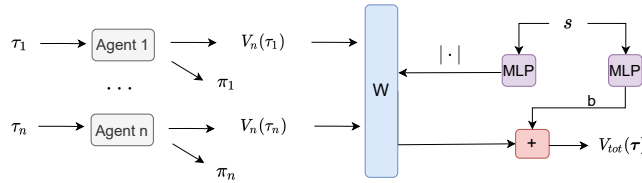
D.7 VMIX

Figure 12: Architecture for VMIX: $|\cdot|$ denotes **absolute value operation**, decomposing V_{tot} into V_i.

VMIX [8] [27] combines the Advantage Actor-Critic (A2C) [26] with QMIX to extend the monotonicity constraint to value networks (not Q value network), as shown in Eq. 19 and Figure 12. We verified that the monotonicity constraint also has a positive effect on the value network based on VMIX (Figure 10).

$$\begin{aligned} V_{tot}(s; \boldsymbol{\theta}, \phi) \\ = g_{\phi} \left(s, V^1 \left(\tau^1; \theta^1\right), \dots, V^N \left(\tau^N; \theta^N\right)\right) \end{aligned} \tag{18}$$

---

[7]LICA code: https://github.com/mzho7212/LICA
[8]VMIX code: https://github.com/hahayonghuming/VDACs

$$\frac{\partial V_{tot}}{\partial V^i} \geq 0, \quad \forall i \in \{1, \ldots, N\} \tag{19}$$

where ϕ is the parameter of value mixing network, and θ_i is the parameter of agent network. With the centralized value function V_{tot}, the policy networks can be trained by policy gradient (Eq. 20),

$$\hat{g}_i = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_{\theta^i} \left(u_t^i \mid \tau_t^i \right) \Big|_{\theta^i} \hat{A}_t \tag{20}$$

where $\hat{A}_t = r + V_{tot}(s') - V_{tot}(s)$ is the advantage function, and $\mathcal{D}$ denotes replay buffer.

D.8   RELATIONSHIP BETWEEN PREVIOUS WORKS

VDNs requires a linear decomposition of Q values, so it has the strongest monotonicity constraint. Since the weights calculated by softmax (attention mechanism) are greater than or equal to zero, the constraint strengths of Qatten and QMIX are approximately equal. QPLEX just shifts the constraint to advantage values without removing it. WQMIX relaxes the monotonicity constraint even further by a true Q value network and theoretical guarantees. LICA completely removes the monotonicity constraint by new network architecture. We rank the strength of the monotonicity constraints on these MARL algorithms:

$$\text{VDNs} > \text{QMIX} \approx \text{Qatten} > \text{QPLEX} > \text{WQMIX} > \text{LICA} \tag{21}$$