RETHINKING EXPLORATION AND EXPERIENCE EXPLOITATION IN VALUE-BASED MULTI-AGENT REINFORCEMENT LEARNING

Anatolii Borzilov
MIPT, FRC CSC RAS
Moscow, Russia
borzilov.av@gmail.com

Alexey Skrynnik
AIRI, FRC CSC RAS
Moscow, Russia
skrynnik@airi.net

Aleksandr Panov
AIRI, MIPT
Moscow, Russia
panov@airi.net

ABSTRACT

Cooperative Multi-Agent Reinforcement Learning (MARL) focuses on developing strategies to effectively train multiple agents to learn and adapt policies collaboratively. Despite being a relatively new area of study, most MARL methods are grounded in well-established approaches used in single-agent deep learning tasks due to their proven effectiveness. In this paper, we focus on the exploration problem inherent to many MARL algorithms. These algorithms frequently introduce new hyperparameters and incorporate auxiliary components, like additional models, complicating the adaptation process of the underlying RL algorithm to better fit multi-agent environments. We aim to optimize a deep MARL algorithm with minimal modifications to the renowned QMIX approach. Our investigation into the exploitation-exploration dilemma reveals that the performance of cutting-edge MARL algorithms can be equaled by merely tweaking the ϵ-greedy policy. This modification depends on the ratio of available joint actions to the number of agents. Moreover, we enhance the training aspect of the replay buffer to decorrelate experiences based on recurrent rollouts rather than episodes. The improved algorithm is not only easy to implement but also aligns with state-of-the-art methods without adding significant complexity.

1 INTRODUCTION

Multi-Agent Reinforcement Learning (MARL) represents an emerging field within artificial intelligence, where the goal is to develop robust strategies for training multiple agents to collaboratively learn and adapt their policies. Notable examples of MARL's success include its application in complex tasks such as autonomous driving, where multiple vehicles must coordinate to navigate, as demonstrated by the Nocturne framework (Vinitsky et al., 2022). Additionally, IMP-MARL provides a platform for evaluating the scalability of cooperative MARL methods, which are responsible for planning inspections and repairs of specific system components with the goal of minimizing maintenance costs (Leroy et al., 2024). MATE addresses target coverage control challenges in real-world scenarios, presenting an asymmetric cooperative-competitive game featuring two groups of learning agents, cameras and targets, each with opposing goals (Pan et al., 2022). These successes highlight MARL's potential to solve real-world problems that require coordinated actions among multiple agents.

Despite these successes, even in tasks that are close to real-world applications, MARL algorithms are often tested in game-like environments. For example, the SMAC (StarCraft Multi-agent Challenge) and SMAC-v2 simulators are based on the StarCraft II strategy game, where teams of agents cooperate to defeat enemy groups (Samvelyan et al., 2019; Ellis et al., 2024). Similarly, research in Google Football demonstrates MARL's applicability to complex, dynamic tasks (Song et al., 2023). Although relatively new, MARL methodologies often leverage foundational techniques from single-agent deep reinforcement learning tasks due to their established success.

In our paper, we focus on the popular value-based method called QMIX (Rashid et al., 2020b). This method has become the theoretical and technical foundation for many MARL approaches (Zhang et al., 2023; Wang et al., 2020). First, we investigate how the exploration strategy of value-based methods can be improved and propose a simple approach based on the number of available joint actions. Second, we examine the implementation of experience exploitation, which is often not well-articulated in the literature. We address the question of how to properly sample collected data from the replay buffer for methods that use recurrent networks.

To summarize, we make the following **contributions**:

- We propose a novel exploration strategy for value-based methods, which leverages the number of available joint actions, improving the exploration-exploitation trade-off.
- We analyze the under-explored area of experience exploitation in value-based MARL methods, specifically focusing on how to effectively sample data from the replay buffer when recurrent networks are used.
- We extensively study our proposed modifications on two benchmarks, SMAC and POGEMA, demonstrating that our approach achieves comparable or even superior results to state-of-the-art value-based MARL methods.

The paper is organized as follows: Section 2 outlines the background of multi-agent reinforcement learning field, Section 3 provides a review of related literature, Section 4 describes the methodology, and Section 5 details the experimental setup and results.

2 BACKGROUND

This paper addresses the problem of multi-agent cooperative tasks, formalized as decentralized partially observable Markov decision process (Dec-POMDP) tuple $G = \langle S, A, U, P, r, Z, O, n, \gamma \rangle$. State $s \in S$ describes the complete state of the environment at the moment. At each timestep each agent $a \in A \equiv 1, ..., n$ chooses an action $u^a \in U$; chosen actions of all agents form a joint action $\mathbf{u} \in \mathbf{U} \equiv U^n$. These actions result in environment transition to a new state according to the transition function $P(s'|s, u) : S \times U \times S \to [0, 1]$, at each timestep t. The rewards are given according to the reward function $r(s, \mathbf{u}) : S \times S \to \mathbf{R}$, which is shared by all agents, and $\gamma \in [0, 1)$ is a discount factor.

At each timestep, each agent a receives an individual observation $z^a \in Z$ according to the observation function $O(s, a) : S \times A \to Z$. Each agent maintains an action-observation history $\tau^a \in T \equiv (Z \times U)^*$, on which the agent's policy $\pi^a(u^a|\tau^a) : T \times U \to [0, 1]$ is conditioned. The joint policy π associated with an action-value function $Q^\pi(s_t, \mathbf{u}_t) = E_{s_t+1:\infty, \mathbf{u}_t+1:\infty}[R_t|s_t, \mathbf{u}_t]$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ represents the discounted return. The training objective is to find the optimal action-value function.

DQN (Mnih et al., 2015) is a popular algorithm for single-agent tasks, which learns agent's action-value function. For multi-agent tasks, we learn the joint action-value function $Q_{tot}(\boldsymbol{\tau}_t, \mathbf{u}_t, \theta)$, where $\boldsymbol{\tau} \in \mathbf{T}$ is a joint action-observation history and θ are network parameters. During learning, the replay buffer D, consisted of tuples $(\boldsymbol{\tau}_t, \mathbf{u}_t, r_t, \boldsymbol{\tau}_{t+1})$, is utilized. Thus, the network parameters θ will be learned by the TD error:

$$L(\theta) = \mathbf{E}_{(\boldsymbol{\tau}_t, \mathbf{u}_t, r_t, \boldsymbol{\tau}_{t+1}) \sim D} \left[r + \gamma \max_{\mathbf{u}_{t+1}} Q_{tot}(\boldsymbol{\tau}_{t+1}, \mathbf{a}_{t+1}, \theta^-) \right]^2, \tag{1}$$

where θ^- are the parameters of the target network that are periodically updated with θ.

One of the core issues of cooperative MARL is that simultaneous learning of multiple agents induces non-stationarity of the environment. That leads to the problem that decentralized learning of multiple agents is unstable. As a solution, the centralized training with decentralized execution (CTDE) approach was introduced. According to this approach, the execution is decentralized, which means that each agent a chooses actions according to its local action-observation history τ^a. Despite that, the training is centralized, and during training the learning algorithm has the access to the state s

of the environment. In order to allow each agent a to participate in a decentralized execution, it is important to assert that:

$$\arg\max_{\mathbf{u}} Q^{\pi}(s, \mathbf{u}) = (\arg\max_{u^1} Q_1(\tau^1, u^1) ... \arg\max_{u^n} Q_n(\tau^n, u^n)) \tag{2}$$

One of the polular methods to solve the problem is QMIX (Rashid et al., 2020b). This method is a variant of DQN (Mnih et al., 2015) for multiagent tasks, and is based on the ideas of VDN (Sunehag et al., 2018). QMIX uses the mixing network during training to enforce (2). Weights of the mixing network are produced by a hypernetwork based on the current state s, and are non-negative. The mixing network learns the joint action-value function $Q_{tot}(\boldsymbol{\tau}, \mathbf{u})$. The usage of non-negative weights of the mixing networks enforces that $\frac{\partial Q_{tot}}{\partial Q_a} >= 0, \forall a \in A$, which also guarantees (2).

3 Related Work

The exploration-exploitation dilemma in MARL is closely related to similar challenges in deep RL. Many techniques originally developed for single-agent settings have been adapted for MARL. For instance, curiosity-driven exploration, a method that enhances the exploration process, has been effectively integrated into MARL to manage complexities arising from multiple interacting agents. Additionally, tools such as replay buffers and recurrent neural networks are employed to better handle and utilize data collected during agent interactions. However, while these adaptations improve exploration and data utilization, they do not directly address the non-stationarity problem inherent in MARL. Below, we provide an overview of such techniques in single-agent RL and their application in MARL, highlighting their strengths and limitations in the multi-agent context.

3.1 Exploration in MARL.

The exploration problem is a well-studied topic in reinforcement learning. Bootstrapped DQN (Osband et al., 2016) learns several separate Q-value functions, and at the beginning of each episode samples one of these functions. Then, the agent follows the greedy policy for that sampled function. This way, the method allows the agent to use temporally-extended exploration during the whole episode. ϵz-greedy (Dabney et al., 2020) modifies the ϵ-greedy method, and instead of sampling single actions, it samples options of actions, which agent follows for the number of steps that is sampled according to the distribution z. Another approach is to use an intrinsic reward to direct the exploration process. ICM (Pathak et al., 2017) adopts an inverse model to extract features out of inputs that ignore uncontrollable aspects of the environment, and then uses the prediction error of these features as an intrinsic reward. VIME (Houthooft et al., 2016) uses Bayesian neural networks to approximate environment dynamics and then maximizes the information gain about the agent's belief of environment dynamics. VDM (Bai et al., 2021) models the stochasticity in dynamics to enhance predictions and computes the intrinsic reward using the environmental state-action transitions probabilities. RND (Burda et al., 2018) computes the exploration bonus based on state novelty, which is estimated by distilling a fixed randomly initialized network into another one. SMM (Lee et al., 2019) learns a policy to match its state marginal distribution with a target state distribution. NGU (Badia et al., 2019) computes intrinsic reward based on two components: exploration bonus for lifelong novelty, which is computed using RND, and episodic novelty bonus. To compute the episodic novelty bonus, it uses episodic memory, which contains all the visited states in the current episode. Then, it encourages the agent to visit as many different states as possible during a single episode. Agent57 (Badia et al., 2020), being based on NGU, also learns a family of policies with different degrees of exploration and exploitation. It uses an adaptive meta-controller to choose from these policies, which allows to control the intensity of exploration during the training process. Instead of exploring novel states, SMiRL (Berseth et al., 2020) tries to minimize a surprise from new states, thus developing behavior that decreases entropy. Such an approach allows the learning agent to develop meaningful skills in unstable environments, where unexpected events happen on their own.

Generally, multi-agent methods try to adopt existing single-agent approaches for exploration. LIIR (Du et al., 2019) uses an individual intrinsic reward for each agent, which allows the agents to be stimulated differently. The parameters of intrinsic rewards are learned using the centralized

critic to maximize the team reward. EMC (Zheng et al., 2021) utilizes a curiosity module, which is trained to predict individual Q-values of agents. These prediction errors are used as additional intrinsic rewards. Wang et al. (Wang et al., 2019) introduce two methods that are based on measuring of the interactions between agents to compute intrinsic rewards: EITI uses the mutual information between agents' trajectories, and EDTI quantifies the influence of an agent on expected returns of other agents. MAVEN (Mahajan et al., 2019) uses a latent variable, which is generated by a hierarchical policy, to perform coordinated exploration in different modes. Then, MAVEN maximizes the mutual information between the observed trajectories to achieve diverse behavior. CMAE (Liu et al., 2021) utilizes restricted space exploration and shared goals. It first explores goals from a low-dimensional restricted space and then trains exploration policies to reach these goals, which represent under-explored states. This method showed significant improvement in sparse-reward environments. SMMAE (Zhang et al., 2023) enhances exploration in two different ways. Firstly, it introduces an intrinsic reward based on SMM. Secondly, it uses adaptive exploration, and bases each agent's probabilities of choosing random actions on correlation between agents. It predicts the actions of each agent based on other agents' observations to measure the correlation between them and increases the probability of choosing random actions if the correlation is too high.

3.2 Experience Exploitation in MARL

While classic algorithms during training process replay whole episode sequences, it may create number of practical issues because of varying episode length and correlated states in trajectories. To solve this issue, R2D2 (Kapturowski et al., 2018) trains on sequences of transitions of fixed length, which overlap by half of their length and never cross boundaries of the episode. Though that method allows to overcome some issues, which are created by learning on the whole episodes, it also creates an issue of necessity to properly initialize hidden recurrent states during training. R2D2 introduces two strategies to solve this issue: storing the recurrent state in replay buffer, and using "burn-in" phase during training, i.e. using the first half of the training sequence only for initialization of the recurrent states, and apply the training objective to the second half of the sequence.

Number of methods also utilizes prioritized experience replay (Schaul et al., 2015). Ape-X (Horgan et al., 2018) suggests to use the absolute TD error for experience priorities. R2D2 (Kapturowski et al., 2018) and R2D3 (Paine et al., 2019) use the mixture of the maximum absolute TD error and the mean TD error in the sequence.

Most works focused on modifying experience replay consider single-agent domain, though some works adopt this concept for multi-agent tasks. MAC-PO (Mei et al., 2023) only uses weights for weighted error, and sample training transitions with uniform distribution. (Wang & Zhang, 2019) adopts prioritized for multi-agent task and sets priorities for each transition – and trains on mini-batch of transitions without recurrency. Number of methods, like QMIX (Rashid et al., 2020b) and its derivatives (Mahajan et al., 2019; Wang et al., 2020; Zhang et al., 2023), sample for training uniformly full episodes . So far, by our knowledge, there are no works considering modifying experience replay so that fixed-length sequences would be used for training, with or without prioritization.

4 Method

This section outlines the key methodological advancements introduced in our research to enhance exploration and training efficiency in reinforcement learning. We first present a novel modification to the traditional ϵ-greedy policy, where the exploration probability is dynamically adjusted based on the count of available joint actions.

Following this, we describe our enhanced replay buffer strategy designed to improve the training process's efficiency and stability. Instead of relying on full episodes, we utilize overlapping sequences of fixed length, allowing for more effective learning across episode boundaries.

Modification of ϵ-greedy policy. As ϵ-greedy policy uses a constant value of a probability of choosing random actions ϵ, it may be hard to adapt the exploration degree to the current environment situation. The number of available to agents actions may vary in some environments, and the number of agents may change during the episode as well. The varying number of the available joint actions

leads to the necessity of dynamically adaptation of the exploration extent in order to properly explore environment states.

In contrast to ϵ-greedy policy, where a constant value of ϵ is used, we compute the exploration probability ϵ using the available actions count. We assume that the more joint actions $\mathbf{U}$ are available, the more intense exploration it is required to find the optimal strategy. According to that reasoning, we introduce the following way to compute the value of ϵ_t:

$$\epsilon_t = tanh(\alpha \cdot \sqrt{log(|\mathbf{U}_t|)}) \tag{3}$$

where $|\mathbf{U}_t|$ is a count of the available joint actions at the step t; α is a constant hyperparameter.

Also, we set minimum and maximum boundaries ϵ_{min} and ϵ_{max}, so that ϵ_t would always stay in reasonable limits.

Scaling the value of ϵ_t on available joint actions count may allow adapt exploration intensity to the current state. In some environments, like SMAC (Samvelyan et al., 2019), number of actions, available to agents, may greatly vary.

Replay buffer enhancement. Replay buffer $\mathcal{D}$ consists of a fixed number of episodes' steps D. Instead of training on full episodes, we train using sequences of steps of fixed length m, in order to decrease the dependency of the learning process on the episodes length. These sequences aren't restricted by episodes boundaries and may contain steps of different episodes. At the beginning of each train iteration, recurrent state is initialized to zero. The first half of each sequence is used for the initialization of the recurrent state, and the training objective is only applied to the second half of a sequence. If a sequence contains parts of different episodes, at the step of switching between episodes the recurrent state is zero initialized again. Also, to decrease dependency on the environment, we run train iterations after the constant number of rollout time-steps performed.

The process of insertion of data in replay buffer is described in Algorithm 1. As we sample a new episode of length T, we need to store it in replay buffer $\mathcal{D}$. The maximum amount of transitions that we store in $\mathcal{D}$ is D, so, if the size of $\mathcal{D}$ exceeds that limit, we remove the oldest transitions.

Algorithm 1: Inserting transitions in replay buffer

Input: List of transitions $\mathcal{D}$, buffer size D
Output: List of transitions $\mathcal{D}$

Sample transition tuples $\rho \leftarrow \left\{\left(s_t, r_t \left\{\left(z_t^a, u_t^a, z_{t+1}^a\right) | a = 1, ..., n\right\}\right) | t = 0, ..., T-1\right\}$
for *each step* $t = 0, ..., T-1$ **do**
  if *size*$(\mathcal{D}) = D$ **then**
    $\mathcal{D} \leftarrow \mathcal{D}[1:]$ `// Pop oldest index`
  end
  $\mathcal{D} \leftarrow$ concat$(\mathcal{D}, \rho_t)$
end

We sample training sequences following the Algorithm 2. As we train the network on batches of size B, we uniformly choose B starting indices. Then, for each sampled index i we put in batch $\mathcal{B}$ a sequence which consists of m transitions starting from index i up to index $i + m$. If the value $i + m$ exceeds the capacity of $\mathcal{D}$, we select in the sequence transitions from i up to the last transition stored in $\mathcal{D}$, and select the rest of transitions starting from the index 0 to fill the sequence up to the size m.

5 EXPERIMENTS

In this section, we present our experimental results on SMAC (Samvelyan et al., 2019) and POGEMA (Skrynnik et al., 2024a) benchmarks. As we study the effectiveness of QMIX (Rashid et al., 2020b) method with the proposed modifications of the exploration policy and replay buffer, we consider the version of QMIX with both of the modifications, QMIX with only replay buffer modi-

Algorithm 2: Sample transitions from replay buffer

Input: List of transitions $\mathcal{D}$, sequence size m, batch size B
Output: Batch of transitions $\mathcal{B}$

$\mathcal{B} \leftarrow ()$ `// Initialize batch as an empty list`
while *size*$(\mathcal{B}) < B$ **do**
  $i \sim \mathcal{U}(0, \text{size}(\mathcal{D}) - 1)$ `// Randomly sample starting index of a sequence`
  if $i + m <$ *size*$(\mathcal{D})$ **then**
    $b \leftarrow \mathcal{D}[i : i + m]$
  else
    $b \leftarrow \text{concat}(\mathcal{D}[i :], \mathcal{D}[: \text{size}(\mathcal{D}) - i])$
  end
  $\mathcal{B} \leftarrow \text{concat}(\mathcal{B}, b)$
end

fication, and QMIX with only exploration policy modification. The source code of these methods is available at [1].

5.1 Comparison on SMAC

To study the impact of the modified exploration policy and replay buffer modification, we experiment on different SMAC (Samvelyan et al., 2019) scenarious, and compare results with QMIX (Rashid et al., 2020b) and SMMAE (Zhang et al., 2023). Here, SMMAE is a specialized approach that enhances the exploration abilities of QMIX.

We also conduct ablation experiments to study the impact of each algorithm modifications separately. SMAC benchmark is focused on micromanagement task of the popular game StarCraft II, where each unit is controlled by a different agent in order to defeat the opponent army controlled by a game's build-in scripted AI. The game is considered won if agents managed to kill every enemy unit within the time limit, and the quality metrics is win rate. Initially, following SMMAE Zhang et al. (2023), we selected three scenarios for the experiments on SMAC: *6h_vs_8z*, *2c_vs_64zg*, and *corridor*. However, for *6h_vs_8z*, we observed that the agent learned to exploit the reward system. The enemies had shields that regenerated over time, and under the standard settings in SMAC, agents were rewarded for regeneration of enemies' shields as if it were damage. As a result, it was more advantageous for agents to damage the shields and then retreat out of the enemies' line of sight, which lead to low win rate. As it was a known issue, which wasn't planned to be fixed[2], we decided to replace the map *6h_vs_8z* with a map *MMM2*, which is also considered to be "super-hard". Figure 1 includes screenshots of the selected maps. The version of StarCraft II used for the evaluation is SC2.4.10 (B75689).

Among the chosen scenarios the total number of actions of each agent varied from 18 to 70, and number of agents varied from 2 to 10. Given that, the maximum theoretical amount of unique joint actions is 18^{10}, though in actual experiments lots of actions are often unavailable.

Following (Zhang et al., 2023), we use in our experiments QMIX with Adam (Kingma, 2014) optimizer with default hyperparameters. ϵ anneal time is 50000 steps, and the exploration hyperparameter α is set to 0.04 for *corridor* and 0.02 for *2c_vs_64zg* and *MMM2*. A detailed implementation description is provided in Appendix B.

The results of the experiments on SMAC are shown in Figure 2. On a super-hard scenario *corridor* poor QMIX results compared to other algorithms indicate that additional exploration significantly increases learning performance. SMMAE and QMIX with adaptive ϵ achieve similar results, and QMIX with both modifications has slightly better performance. On a hard scenario *2c_vs_64zg* a replay buffer modification results in worse learning performance, meanwhile QMIX, QMIX with modified exploration policy and SMMAE achieve the similar results. On a super-hard scenario *MMM2*

[1] https://github.com/tolyan3212/re-qmix
[2] https://github.com/oxwhirl/smac/issues/72

(a) corridor (b) 2c_vs_64zg (c) MMM2

Figure 1: Screenshot examples of SMAC scenarios from the StarCraft II game, which are used in experiments. In all scenarios, the red units are controlled by RL agents, while the blue units are controlled by the game's built-in AI. The RL agents are trained jointly during learning but make independent decisions during testing, following the centralized training, decentralized execution paradigm. In the *corridor* scenario, the RL-controlled zealots must coordinate their movement towards the lower left corner of the map, where a narrow corridor is located. This positioning allows them to defeat a large number of zerg units. In the *2c_vs_64zg* scenario, the RL agents control two Colossus, exploiting the units' ability to traverse high ground to defeat a large number of zergs. In the *MMM2* scenario, a group of 2 Marauders, 7 Marines, and 1 Medivac, controlled by RL agents, attempt to defeat a larger group consisting of 3 Marauders, 8 Marines, and 1 Medivac.

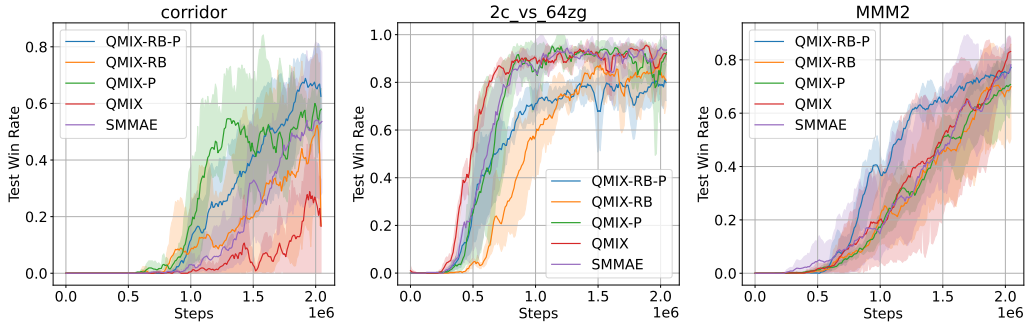


Figure 2: Comparison of the mean test win rate of proposed modifications and other MARL algorithms on SMAC. QMIX-RB-P stands for QMIX with replay buffer and exploration policy modifications; QMIX-RB stands for QMIX with replay buffer modification, and QMIX-P stands for QMIX with exploration policy modification. Plots show the mean and 95% confidence interval across five runs. For the corridor scenario, algorithms with enhanced exploration – QMIX-P, QMIX-RB-P and SMMAE – have better performance compared with QMIX; replay buffer modification also improves the results, and the best performance is achieved by QMIX-RB-P. For the 2c_vs_64zg scenario, replay buffer modification leads to worse performance, and the other algorithms have similar results. For the MMM2 scenario, QMIX-RB-P has the steepest learning curve, but the final performance of the algorithms is almost identical.

an algorithm with both exploration policy and replay buffer modifications has slightly steeper learning curve, though the final winning rates of algorithms are almost identical.

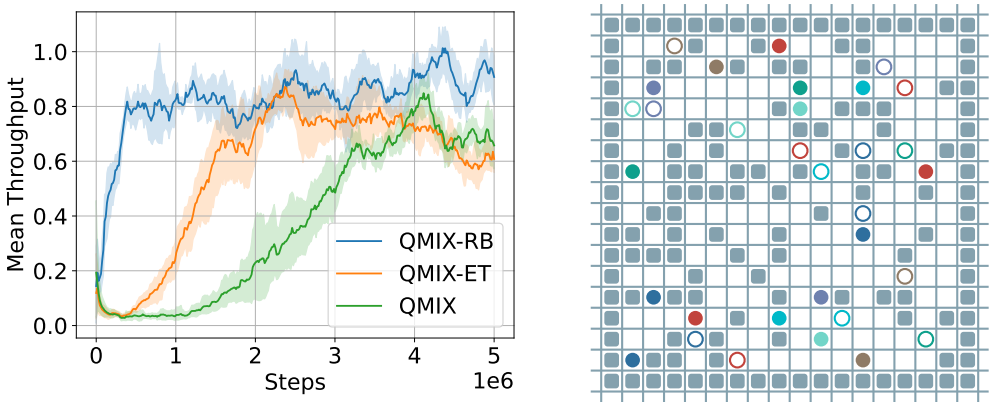
Comparison of the proposed algorithm with SMMAE, wich uses additional attention-based and VAE modules to enhance exploration, shows that it's possible to achieve the similar exploration effectiveness with a simple in terms of computation and implementation modification of ϵ-greedy policy. We also conducted additional experiments with state-of-the-art MARL algorithms, QPLEX(Wang et al., 2020) and WQMIX(Rashid et al., 2020a), the results of which are presented in Appendix A.

5.2 Comparison on POGEMA

POGEMA (Skrynnik et al., 2024a) is a grid-like multi-agent pathfinding environment, where multiple agents are supposed to move to their goals in order to get a collective reward, which is given

when any of agent steps on a goal. Additional small rewards are given when agents shorten the distance to their goals, as in (Skrynnik et al., 2024b). The task of decentralized multi-agent pathfinding is particularly challenging, as highlighted by several specialized methods (Andreychuk et al., 2024; Wang et al., 2023; Skrynnik et al., 2024c; 2023; Sartoretti et al., 2019).

We consider the LifeLong scenario, where when an agent accomplishes its goal, a new goal is set for it. There are obstacles present on the map, and agents cannot pass through a cell occupied by another agent, which necessitates adopting cooperative behavior to maximize rewards. We use random generated maps for training, which means that the agents' training goal is not to memorize the map, but to be able to adapt to a new layout and find the way to the goal on an unknown map. The quality metric used is throughput, i.e., the ratio of the number of the accomplished goals (by all agents) to the episode length.



(a) POGEMA results.

(b) POGEMA environment.

Figure 3: (a) Comparison of the mean average throughput on POGEMA with large episodes. QMIX-RB stands for QMIX with replay buffer modification; QMIX-ET stands for QMIX with early training start. Plots show the mean and 95% confidence interval across five runs. QMIX-RB starts quickly improving its performance from the beginning of training, achieving the best results. QMIX-ET has a steeper learning curve compared to QMIX, but it still takes a lot of time to achieve competitive results. Please note that the average throughput can exceed 1.0. (b) Illustration of a random POGEMA map with a size of 16×16 and a population of 16 agents. The agents are represented as colored filled circles, and their targets are shown as circles of the same color. Each agent has a single, unique target.

To further study the impact of the replay buffer modification, we conduct experiments on POGEMA environment with large episode length of 1000 steps and with batch size 64. The goal of that experiment is to simulate the situation, when the environment's episodes are very large and contain an amount of information which is hard to process during the training. We compared QMIX with proposed replay buffer modification with other two versions of QMIX: one is a usual QMIX implementation, where training starts when enough episodes are sampled to form a full batch of the given size. The second version of QMIX has no restrictions on the start of training, so that training starts right after the first episode was sampled, but with each next sampled episode the batch size is increased up to 64.

For this experiment, we do not consider the exploration policy modification as it depends on the current number of available actions, which is constant for POGEMA environment – this means that the usage of the modified exploration policy would still result in a constant value of ϵ.

The results of that comparison are shown in Figure 3. While QMIX with early training start has steeper learning curve than QMIX, the results of QMIX with modified replay buffer are significantly superior. These results indicate that proposed replay buffer modification may decrease the dependency on the environment and it's episodes length, which simplifies the selection of some hyperparameters such as batch size.

6 Conclusion and limitations

In this paper, we have explored the impact of a modified exploration policy and replay buffer modification in the context of cooperative MARL, using the SMAC and POGEMA environments as benchmarks. Our results demonstrate that these modifications can significantly enhance the performance of the QMIX algorithm without introducing substantial complexity. Our enhancements offer a streamlined alternative to complex MARL methods, achieving results comparable to state-of-the-art methods with minimal alterations to the original algorithm, thereby simplifying the adaptation process for diverse multi-agent environments.

A potential limitation of our work is the lack of formal theoretical guarantees regarding the exploration component. This could be addressed by framing the exploration process as a multi-armed bandit problem, where the number of available arms changes at each time step. While this theoretical perspective could provide more rigorous guarantees, fully developing it within the scope of this paper would be challenging.

References

Anton Andreychuk, Konstantin Yakovlev, Aleksandr Panov, and Alexey Skrynnik. Mapf-gpt: Imitation learning for multi-agent pathfinding at scale. *arXiv preprint arXiv:2409.00134*, 2024.

Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martin Arjovsky, Alexander Pritzel, Andrew Bolt, et al. Never give up: Learning directed exploration strategies. In *International Conference on Learning Representations*, 2019.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *International conference on machine learning*, pp. 507–517. PMLR, 2020.

Chenjia Bai, Peng Liu, Kaiyu Liu, Lingxiao Wang, Yingnan Zhao, Lei Han, and Zhaoran Wang. Variational dynamic for self-supervised exploration in deep reinforcement learning. *IEEE Transactions on neural networks and learning systems*, 2021.

Glen Berseth, Daniel Geng, Coline Manon Devin, Nicholas Rhinehart, Chelsea Finn, Dinesh Jayaraman, and Sergey Levine. Smirl: Surprise minimizing reinforcement learning in unstable environments. In *International Conference on Learning Representations*, 2020.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2018.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Will Dabney, Georg Ostrovski, and André Barreto. Temporally-extended {\epsilon}-greedy exploration. *arXiv preprint arXiv:2006.01782*, 2020.

Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Foerster, and Shimon Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. *Advances in neural information processing systems*, 29, 2016.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019.

Pascal Leroy, Pablo G Morato, Jonathan Pisane, Athanasios Kolios, and Damien Ernst. Imp-marl: a suite of environments for large-scale infrastructure management planning via marl. *Advances in Neural Information Processing Systems*, 36, 2024.

Iou-Jen Liu, Unnat Jain, Raymond A Yeh, and Alexander Schwing. Cooperative exploration for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*, pp. 6826–6836. PMLR, 2021.

Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. *Advances in neural information processing systems*, 32, 2019.

Yongsheng Mei, Hanhan Zhou, Tian Lan, Guru Venkataramani, and Peng Wei. Mac-po: Multi-agent experience replay via collective priority optimization. *arXiv preprint arXiv:2302.10418*, 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

Tom Le Paine, Caglar Gulcehre, Bobak Shahriari, Misha Denil, Matt Hoffman, Hubert Soyer, Richard Tanburn, Steven Kapturowski, Neil Rabinowitz, Duncan Williams, et al. Making efficient use of demonstrations to solve hard exploration problems. *arXiv preprint arXiv:1909.01387*, 2019.

Xuehai Pan, Mickel Liu, Fangwei Zhong, Yaodong Yang, Song-Chun Zhu, and Yizhou Wang. Mate: Benchmarking multi-agent reinforcement learning in distributed target coverage control. *Advances in Neural Information Processing Systems*, 35:27862–27879, 2022.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

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

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284, 2020b.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Alexey Skrynnik, Anton Andreychuk, Konstantin Yakovlev, and Aleksandr I Panov. When to switch: planning and learning for partially observable multi-agent pathfinding. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

Alexey Skrynnik, Anton Andreychuk, Anatolii Borzilov, Alexander Chernyavskiy, Konstantin Yakovlev, and Aleksandr Panov. Pogema: A benchmark platform for cooperative multi-agent navigation. *arXiv preprint arXiv:2407.14931*, 2024a.

Alexey Skrynnik, Anton Andreychuk, Maria Nesterova, Konstantin Yakovlev, and Aleksandr Panov. Learn to follow: Decentralized lifelong multi-agent pathfinding via planning and learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17541–17549, 2024b.

Alexey Skrynnik, Anton Andreychuk, Konstantin Yakovlev, and Aleksandr Panov. Decentralized monte carlo tree search for partially observable multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17531–17540, 2024c.

Yan Song, He Jiang, Haifeng Zhang, Zheng Tian, Weinan Zhang, and Jun Wang. Boosting studies of multi-agent reinforcement learning on google research football environment: the past, present, and future. *arXiv preprint arXiv:2309.12951*, 2023.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 2018 International Conference on Autonomous Agents and Multiagent Systems*, volume 3, pp. 2085–2087. ASSOC COMPUTING MACHINERY, 2018.

Tijmen Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26, 2012.

Eugene Vinitsky, Nathan Lichtlé, Xiaomeng Yang, Brandon Amos, and Jakob Foerster. Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world. *Advances in Neural Information Processing Systems*, 35:3962–3974, 2022.

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. In *International Conference on Learning Representations*, 2020.

Tonghan Wang, Jianhao Wang, Yi Wu, and Chongjie Zhang. Influence-based multi-agent exploration. In *International Conference on Learning Representations*, 2019.

Yishen Wang and Zongzhang Zhang. Experience selection in multi-agent deep reinforcement learning. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 864–870. IEEE, 2019.

Yutong Wang, Bairan Xiang, Shinan Huang, and Guillaume Sartoretti. Scrimp: Scalable communication for reinforcement-and imitation-learning-based multi-agent pathfinding. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9301–9308. IEEE, 2023.

Shaowei Zhang, Jiahan Cao, Lei Yuan, Yang Yu, and De-Chuan Zhan. Self-motivated multi-agent exploration. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pp. 476–484, 2023.

Lulu Zheng, Jiarui Chen, Jianhao Wang, Jiamin He, Yujing Hu, Yingfeng Chen, Changjie Fan, Yang Gao, and Chongjie Zhang. Episodic multi-agent reinforcement learning with curiosity-driven exploration. *Advances in Neural Information Processing Systems*, 34:3757–3769, 2021.

A ADDITIONAL SMAC EXPERIMENTS

We also conducted comparison of the proposed modifications with state-of-the-art value-based MARL algorithms, as QPLEX (Wang et al., 2020) and WQMIX (Rashid et al., 2020a), which are not enhance idea of QMIX further in other way.

The results are presented in Figure 4. The implementation of QPLEX algorithm used in experiments was provided by the repository [3], and the implementation of WQMIX was provided by the repository [4].

According to the results, the *corridor* map proved to be challenging for both QPLEX and WQMIX algorithms to solve during the training period. QPLEX showed competitive results on the *2c_vs_64zg* map but underperformed on the *MMM2* scenario. OW-QMIX performed similarly to QMIX-RB-P on *MMM2* and had a steep learning curve on the *2c_vs_64zg* map, although its final results on that map were worse than those of QPLEX and QMIX-P. Overall, CW-QMIX demonstrated slightly lower performance compared to the other algorithms.

In summary, while algorithms such as QPLEX and WQMIX may outperform QMIX with the proposed modifications on certain scenarios, their performance is less stable across various scenarios, and these methods struggle to succeed in more challenging scenarios, such as the corridor.

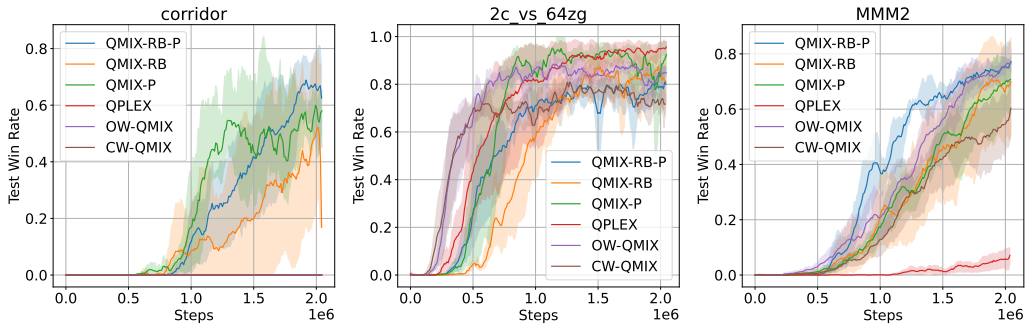


Figure 4: Comparison of the mean test win rate of proposed modifications with state-of-the-art MARL algorithms QPLEX and WQMIX on SMAC. QMIX-RB-P stands for QMIX with replay buffer and exploration policy modifications; QMIX-RB stands for QMIX with replay buffer modification, and QMIX-P stands for QMIX with exploration policy modification. Plots show the mean and 95% confidence interval across five runs. The corridor scenario proved to be too challenging for the QPLEX and WQMIX algorithms. In the 2c_vs_64zg scenario, CW-QMIX performed relatively close to QMIX-RB and QMIX-RB-P, and QMIX-P with QPLEX showed the best results. In the MMM2 scenario, QPLEX showed the worst results, CW-QMIX has a slightly worse performance compared to QMIX-RB and QMIX-P, while QMIX-RB-P and OW-QMIX achieved the best results.

B IMPLEMENTATION DETAILS

We use the QMIX (Rashid et al., 2020b) algorithm as the base method for the proposed modifications. Our implementation is based on PyMARL (Samvelyan et al., 2019). Following SMMAE (Zhang et al., 2023), we changed RMSProp (Tieleman, 2012) optimizer with Adam (Kingma, 2014) optimizer with default hyper-parameters.

At the beginning of training, we linearly anneal ϵ from 1.0 to 0.05 over 50,000 steps. For methods without the replay buffer modification, the buffer size is set to 5,000 episodes, and after sampling a new episode from the environment, we select a batch of 32 episodes for training. For methods with the replay buffer modification, for every 128 steps sampled from the environment, we select a batch of 64 sequences for training. Each sequence is 128 steps long, with 64 steps used for the "burn-in" phase and the remaining 64 steps used for training. In the POGEMA (Skrynnik et al.,

[3] https://github.com/wjh720/QPLEX
[4] https://github.com/hijkzzz/pymarl2

2024a) environment, the replay buffer size is set to 1,000 episodes or 800,000 steps. The agents' network architecture is the same as that of QMIX, with a GRU (Chung et al., 2014) recurrent layer having a 64-dimensional hidden state. The target network is updated every 200 training episodes for methods without the replay buffer modification, and every 10,000 steps for methods with the replay buffer modification. The complete hyper-parameters setup is shown in Table 1.

Table 1: The hyper-parameters of proposed modifications and base version of QMIX algorithm.

Hyper-parameter	Value
Sequence length	128 steps
Length of a burn-in phase	64 steps
ϵ anneal time	50000 steps
ϵ finish	0.05
Learning rate	0.001
α	0.02 (0.04 for *corridor*)
γ	0.99
GRU hidden size	64
Mixing network size	32
Mixer's hypernet layers	2
Mixer's hypernet hidden size	64
Optimizer	Adam
Improved Replay Buffer (w/ RB)	
Size for SMAC	200000 steps
Size for POGEMA	800000 steps
Target update interval	10000 steps
Default Replay Buffer (w/o RB)	
Size for SMAC	5000 episodes
Size for POGEMA	1000 episodes
Target update interval	200 episodes