The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games

Chao Yu*[1] **Akash Velu***[2] **Eugene Vinitsky**†[2] **Yu Wang**[1] **Alexandre Bayen**[2]

Yi Wu†[1,3]

[1] Tsinghua University [2] University of California, Berkeley [3] Shanghai Qi Zhi Institute

Abstract

Proximal Policy Optimization (PPO) is a popular on-policy reinforcement learning algorithm but is significantly less utilized than off-policy learning algorithms in multi-agent settings. This is often due the belief that on-policy methods are significantly less sample efficient than their off-policy counterparts in multi-agent problems. In this work, we investigate Multi-Agent PPO (MAPPO), a variant of PPO which is specialized for multi-agent settings. Using a 1-GPU desktop, we show that MAPPO achieves surprisingly strong performance in three popular multi-agent testbeds: the particle-world environments, the Starcraft multi-agent challenge, and the Hanabi challenge, with minimal hyperparameter tuning and without any domain-specific algorithmic modifications or architectures. In the majority of environments, we find that compared to off-policy baselines, MAPPO achieves strong results while exhibiting comparable sample efficiency. Finally, through ablation studies, we present the implementation and algorithmic factors which are most influential to MAPPO's practical performance.

1 Introduction

With recent advances in deep reinforcement learning (RL), we have witnessed many achievements in building intelligent agents to solve complex multi-agent challenges: AlphaStar achieved professional-player level performance in Starcraft II [34], OpenAI Five defeated the world champion in Dota II [4], and RL was even used to demonstrate emergent, human-like tool-use in a simulated world [2].

These successes were accomplished using distributed on-policy RL algorithms such as IMPALA [9] and PPO [29] but required massive amounts of parallelism and compute; tens of thousands of CPU cores and tens or hundreds of GPUs were utilized to collect and train on an extraordinary volume of training samples. It appears to have almost become a domain consensus that on-policy RL algorithms such as PPO are much less sample efficient than off-policy learning algorithms, and may not be suitable in academic settings with limited computational resources. Hence, recent multi-agent reinforcement learning (MARL) literature has primarily adopted off-policy learning frameworks, such as MADDPG [19] and value-decomposed Q-learning [31, 25]. A variety of algorithmic variants with domain-specific modifications have been proposed within this framework, producing state-of-the-art results on a wide range of multi-agent benchmarks [13, 36].

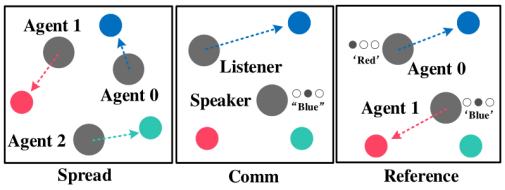
In this paper, we re-examine these conclusions about on-policy policy gradient algorithms by carefully studying the performance of PPO on three popular MARL testbeds: the multi-agent particle-world

*: Equal Contribution

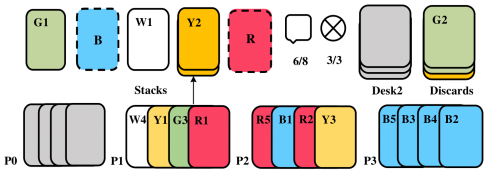
†: Equal Advising

Correspondence to: Akash Velu <akashvelu@berkeley.edu>, Yi Wu <jxwuyi@gmail.com>

Preprint. Under review.



(a) MPE scenarios



(b) 4-player Hanabi-Full

(c) SMAC corridor

(d) SMAC 2c_vs_64zg

Figure 1: Task visualizations. (a) The MPE domain. *Spread* (left): agents need to cover all the landmarks and do not have a color preference for the landmark they navigate to; *Comm* (middle): the listener needs to navigate to a specific landmarks following the instruction from the speaker; *Reference* (right): both agents only know the other's goal landmark and needs to communicate to ensure both agents move to the desired target. (b) The Hanabi domain: 4-player *Hanabi-Full* - figure obtained from (Bard et al., 2020). (c) The *corridor* map in the SMAC domain. (d) The *2c vs. 64zg* map in the SMAC domain.

environment (MPE) [19], the Starcraft multi-agent challenge (SMAC) [26], and the Hanabi challenge [3]. We focus on fully cooperative tasks in order to compare our results directly with the performance of state-of-the-art agents. Additionally, we only utilize a single desktop machine with 1 GPU and 1 multicore CPU for training. [bb=0 0 1280 960]

Despite computational constraints, we find that with simple modifications, PPO achieves performance competitive to many popular off-policy methods on all test scenarios, including three cooperative MPE tasks, 23 SMAC maps and a full-scale Hanabi game. Importantly, we find that in a majority of scenarios, PPO achieves a similar sample efficiency to the off-policy baselines. Through various ablation studies, we study the impact of several specific implementation modifications of the standard PPO algorithm, and offer practical suggestions for strong PPO performance in multi-agent domains. We call PPO with these modifications Multi-Agent PPO (MAPPO).

Our contributions are summarized as follows:

- We show that Multi-Agent PPO (MAPPO), with minimal hyperparameter tuning and without any domain-specific algorithmic changes or architectures, achieves final performances comparable to various off-policy methods on three ubiquitous benchmarks: Particle World, Starcraft and Hanabi.
- We demonstrate MAPPO obtains these strong results while often using a comparable number of samples to many off-policy methods.
- We analyze five implementation and algorithmic factors that govern the practical performance of MAPPO and release our source code at `https://sites.google.com/view/mappo-neurips`.

2 Related Works

MARL algorithms generally fall between two frameworks: centralized and decentralized learning. Centralized methods [5] assume a cooperative game [22] and directly extend single-agent RL algorithms by learning a single policy to produce the joint actions of all agents simultaneously. In decentralized learning [18], each agent optimizes its own reward independently; these methods can tackle general-sum games but may suffer from instability even in simple matrix games [11]. Recent work has developed two lines of research to bridge the gap between these two frameworks: *centralized training and decentralized execution (CTDE)* and *value decomposition (VD)*. CTDE methods such as MADDPG [19] and COMA [10] improve upon decentralized RL by adopting an actor-critic structure

and learning a centralized critic. VD typically represents the joint Q-function as a function of agents' local Q-functions [31, 25, 30] and has been considered a gold standard in MARL. In this work, we analyze MAPPO both as a CTDE algorithm with a centralized value function, and as a decentralized learning algorithm with a decentralized value function.

Despite the early success of on-policy, policy-gradient (PG) RL algorithms such as TRPO in continuous control tasks [7], recent advances in off-policy methods such as SAC [12] have led to a consensus that the latest PG algorithms such as PPO are less sample efficient than off-policy methods in single-agent benchmarks. These conclusions have also been drawn in multi-agent domains: [23] report that multi-agent PG methods such as COMA are outperformed by MADDPG and QMix [25] by a clear margin in both the particle-world environment (MPE) [20] and the Starcraft multi-agent challenge (SMAC) [26]. [37] additionally analyze weaknesses in existing multi-agent PG methods and propose a multi-agent PG method which achieves strong performance in the SMAC benchmark. Note that although [6] empirically notice that fully decentralized independent PPO training (IPPO) can achieve surprisingly high success rates on some specific hard SMAC maps, the cause for this success is unknown, and the overall performance of IPPO remains worse than QMix in their work. Off-policy MARL methods have also been explored in the Hanabi [3] challenge; [13] achieve the best model-free RL results through a variant of Q-learning. In this work, we re-examine these conclusions about on-policy MARL methods by studying PPO, and show that PPO based methods are able to achieve excellent performance on 3 popular cooperative MARL testbeds.

Many existing works have studied the implementation details of PG algorithms in the single-agent continuous control domain. The benefit of advantage normalization for practical PG performance was established in [33], and [15] suggest the usage of large batch sizes to reduce the high variance of PG methods. Other works have additionally investigated the impact of code-level implementation details and critical hyperparemeters to PPO's performance [8, 1].

We consider multi-agent games and investigate factors that are either largely ignored in the existing literature or are completely unique to the multi-agent setting. Although our multi-agent benchmarks have significantly different dynamics from single-agent environments such as MuJoCo [32], we do find a few previous suggestions, such as input normalization, layer normalization, value clipping, orthogonal initialization, and gradient clipping, to be helpful and include them in our implementation.

3 Multi-Agent PPO (MAPPO)

We study decentralized partially observable Markov decision processes (DEC-POMDP) [21] with shared rewards. A DEC-POMDP is defined by $\langle \mathcal{S}, \mathcal{A}, O, R, P, n, \gamma \rangle$. $\mathcal{S}$ is the state space. $\mathcal{A}$ is the shared action space for each agent. $o_i = O(s; i)$ is the local observation for agent i at global state s. $P(s'|s, A)$ denotes the transition probability from S to S' given the joint action $A = (a_1, \ldots, a_n)$ for all n agents. $R(s, A)$ denotes the shared reward function. γ is the discount factor. Since most of the benchmark environments contain homogeneous agents, we utilize parameter sharing: each agent uses a shared policy $\pi_\theta(a_i|o_i)$ parameterized by θ to produce its action a_i from its local observation o_i, and optimizes its discounted accumulated reward $J(\theta) = \mathbb{E}_{a^t, s^t}\left[\sum_t \gamma^t R(s^t, a^t)\right]$.

Multi-Agent PPO follows the algorithmic structure of the single-agent PPO algorithm by learning a policy π_θ and a value function $V_\phi(s)$ for each agent. $V_\phi(s)$ is used for variance reduction and is only utilized during training; hence, it can take extra global information, allowing MAPPO to follow the CTDE structure. The impact of which inputs are provided to the value function is studied in Sec. 3.2.

We maintain two separate networks for π_θ and V_ϕ and follow common practices in PPO implementations, including: *Generalized Advantage Estimation (GAE)* [28] with advantage normalization, observation normalization, gradient clipping, value clipping, layer normalization, ReLU activation with orthogonal initialization, and a large batch size under our 1-GPU constraint. We also perform a limited grid-search over certain hyper-parameters, including network architecture (i.e., MLP or RNN), learning rate, entropy bonus coefficient, and the initialization scale of the final layer in the policy network. Pseudocode and more training and implementation details can be found in Appendix A and D.

From our experiments, we derive five concrete implementation details that we find to be insightful and particularly critical to MAPPO's practical performance: value normalization, value function inputs, training data usage, policy and value clipping, and death masking. We suggest practical

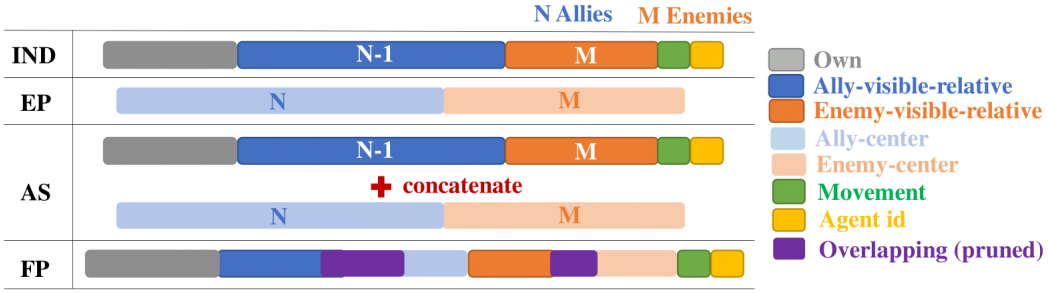


Figure 2: Different value function inputs, with example information contained in each state (specific to SMAC). *IND* refers to using decentralized inputs, i.e., the agents' local observations, *EP* refers to the environment provided global state, *AS* is an agent-specific global state which concatenates *EP* and *IND*, and *FP* refers to a global state in which overlapping features are pruned from *FP*. The *EP* state omits important local data such as agent ID and available actions.

implementation and tuning practices for these aspects of MAPPO and demonstrate the effect of these suggestions through empirical ablation studies presented in Sec. 5.

3.1 Value Normalization

To stabilize value learning, we standardize the targets of the value function by using running estimates of the average and standard deviation of the value targets. This stabilizes the value targets, which, without normalization, can drastically change during training.

Concretely, during value learning, the value network will regress to the normalized target values. When computing the GAE, we will use the running average to denormalize the output of the value network so that the value outputs are properly scaled. We find that using value normalization never hurts training and often significantly improves the final performance of MAPPO.

Suggestion 1: Utilize value normalization to stabilize value learning.

3.2 Input Representation to Value Function

The fundamental difference between many multi-agent CTDE PG algorithms and fully decentralized PG methods is the input to the value network. Therefore, the representation of the value input becomes an important aspect of the overall algorithm. The underlying assumption of centralized value functions is that observing the full global state can make value learning easier. An accurate value function further improves policy learning through variance reduction.

There are two common practices for implementing a centralized value function input: concatenation of all local observations, and an environment-provided global state. In addition to these practices, we investigate several *agent-specific* global state representations. These choices are depicted in Fig. 2.

Concatenation of Local Observations (CL): [19] concatenate all local observations, i.e., $(o_1, \ldots, o_n)$, to form the value function input. While this is generic, the value input dimension can be extremely large when n or the dimension of o_i is large. This makes value learning challenging, which could in turn hurt policy learning. CL also may not contain sufficient global information to reduce a POMDP to an MDP as there can be information which is not observed by any of the agents.

Environment-Provided Global State (EP): Another popular approach is to directly use an environment provided global state. For example, the SMAC environment provides a vector containing information about all agents and enemies, which has been widely adopted as the value function input in most works studying SMAC. However, the environment-provided global state typically only contains information common to all agents, and can omit important local information — this is the case in SMAC, as show in Fig. 2. We hypothesize that the loss of essential local information in the global state is an important factor in yielding the results reported in [6] that decentralized, independent PPO (IPPO), learning purely on local observations, can yield superior results than centralized PPO using the *EP* global state. Our analysis in Sec. 5 will support this hypothesis.

Agent-Specific Global State (AS): To allow value function to leverage global and local information, we propose an *agent-specific* global state: for agent i, we concatenate the environment state s and the local observation o_i. Although AS augments EP with local information, AS can greatly increase the dimension of value network — this is true in SMAC due to overlapping information (see Fig. 2).

Feature-Pruned Agent-Specific Global State (FP): To address the overlap between local and global state, we also evaluate a feature-pruned state representation by removing all the duplicated features in AS. This keeps the same information as AS while significantly reducing the dimensionality.

We generally find that MAPPO performs well when using decentralized value functions or agent-specific global states, indicating that local information can be critical to value learning. MAPPO's performance is strongest with the feature-pruned global state (FP). This ultimately suggests that global information can further improve learning, so long as the state dimension is not high.

Suggestion 2: Include agent-specific features in the global state and check that these features do not make the state dimension substantially higher.

3.3 Training Data Usage

A major trick in PPO is the use of importance sampling to perform off-policy corrections, allowing for sample reuse. After a batch of samples is collected, [29] suggest splitting the large batch into mini-batches and training for multiple epochs. In continuous control domains, the common practice is to train for tens of epochs with around 64 mini-batches per epoch. However, we find that in multi-agent domains, MAPPO's performance degrades when samples are re-used too often. This perhaps could be a result of the non-stationarity issue in MARL. Thus, we use 15 epochs for easy tasks, and 10 or 5 epochs for difficult tasks. Furthermore, similar to the suggestions by [15], we always find that using more data to estimate gradients leads to improved practical performance. Thus, we do not split training data into mini-batches by default. We find that avoiding minibatching is useful in all but one SMAC map, in which splitting the training batch into two mini-batches improves performance. We hypothesize that mini-batching in this scenario helps escape a poor local optimum, similar to phenomena observed in the supervised learning setting [17].

Suggestion 3: Avoid using too many training epochs and do not split data into mini-batches.

3.4 PPO Clipping

Another major trick in PPO is the use of clipped importance ratio and value losses — this attempts to constrain the policy and value functions from drastically changing between iterations. The strength of the clipping is controlled by the ϵ hyperparameter: large ϵ allows greater policy and value changes.

We hypothesize that policy and value clipping, along with the number of training epochs, controls the non-stationarity caused by the changing multi-agent policies. As shown in Sec. 5.4, we observe that while lower ϵ values slows learning speed, they correspond to more consistent policy improvement. On the other hand, higher ϵ values result in larger variance and larger volatility in the performance.

Suggestion 4: For the best PPO performance, tune the clipping ratio ϵ as a trade-off between training stability and fast convergence.

3.5 Death Masking

In multi-agent games, an agent may die before the game terminates (e.g., SMAC). Note that we can always access the global game state to compute an agent-specific global state for those dead agents. Therefore, even if an agent dies and becomes inactive in the middle of a rollout, value learning can still be performed in the following timesteps using inputs containing information of other live agents. This is typical in many existing multi-agent PG implementations.

However, we argue that using these informative states for dead agents during value learning amplifies the bias of the learned value function. Consequently, high value prediction errors in timesteps at which the agent is dead will be accumulated during GAE computation, in turn hindering policy learning over the timesteps in which the agent is alive. Our suggestion is to simply use an agent-specific constant vector, i.e., a zero vector with the agent's ID, as the input to the value function after an agent dies. We call this technique *Death Masking*. Further justifications are provided in Appendix B. We note that this may already be done in existing multi-agent environments, for example, SMAC masks the local observations of dead agents.

Suggestion 5: Use zero states with agent ID as the value input for dead agents.

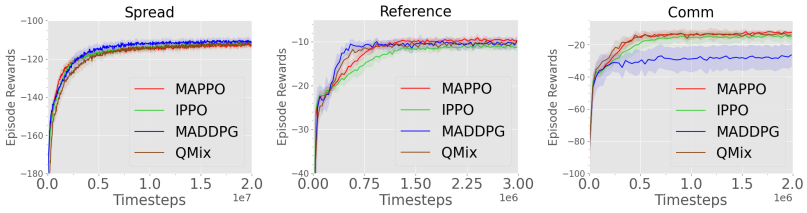


Figure 3: Training curves of different algorithms in the MPEs.

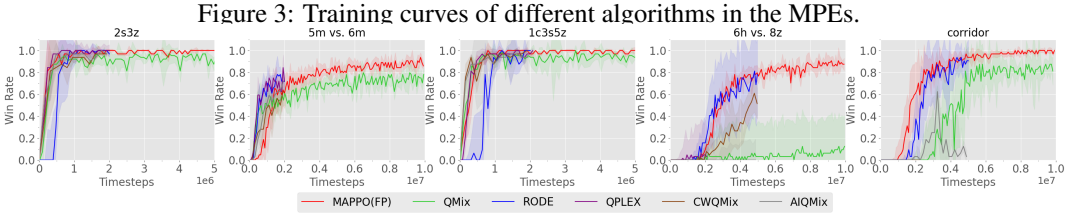


Figure 4: Training curves of different algorithms in SMAC.

4 Main Results

In this section, we compare our implementation of MAPPO with all aforementioned suggestions to other MARL algorithms in the multi-agent particle-world environment (MPE), the Starcraft micromanagement challenge (SMAC), and the Hanabi challenge (Hanabi).

Baseline Comparisons and Experimental Setup: We compare PPO with various value-function inputs to various off-policy baselines, including MADDPG [19] on MPE, QMix [25] and SOTA methods including RODE [36], QPlex [35], CWQMix [24] and AIQMix [16], on SMAC, and SAD [13] on Hanabi. For a fair comparison, we re-implement MADDPG and QMix following the same tuning process as MAPPO over a collection of hyper-parameters such as learning rate, target network update rate, and network architecture. We also specifically test various relevant implementation tricks including value/reward normalization, both hard and soft target network updates for Q-learning, and the input representation to the critic/mixer network. We emphasize that our reported numbers for MADDPG and QMix *all match or exceed the original papers*. Full details can be found in Appendix D.1 and in our open-sourced code. For other baseline methods, we directly obtain the numbers from their original papers or released code.

Each experiment is performed on a desktop machine with 256 GB RAM, one 64-core CPU, and one GeForce RTX 3090 GPU, which is used for forward action computation and training updates.

Empirical Findings: we observe that in the majority of environments, MAPPO achieves results better or comparable to the off-policy comparison methods with ***comparable sample efficiency***.

4.1 MPE Results

We consider the 3 cooperative tasks proposed in [19]: the physical deception task (*Spread*), the simple reference task (*Reference*), and the cooperative communication task (*Comm*), and compare MAPPO with centralized value functions and PPO with decentralized value functions (IPPO) to MADDPG and QMix. As the MPEs do not provide an environment global state, and as the observation dimensions in the MPEs are low, we concatenate all local observations to form the global state. The performance of each algorithm at convergence is shown in Fig. 3. All results are averaged over 10 seeds.

MAPPO achieves performance comparable and even superior to the off-policy benchmarks; we particularly see that MAPPO performs very similarly to QMix on all tasks, and exceeds the performance of MADDPG in the *comm* task, all while using a comparable number of environment steps.

4.2 SMAC Results

In SMAC, we compare MAPPO with several value function inputs (*AS* and *FP*) and IPPO to value-decomposition off-policy methods including QMix, RODE, QPlex, CWQMix and AIQMix. We follow the evaluation metric proposed in [36]: for each seed, we compute the win rate over 32 test games after each training iteration and take the median of the final 10 evaluations as the performance for each training seed. We measure the median success rates over 6 seeds in Table 1, which compares the PPO-based methods to QMix and RODE. Comparisons to all off-policy baselines can be found in

Map	MAPPO(FP)	MAPPO(AS)	IPPO	QMix	RODE*	MAPPO*(FP)	MAPPO*(AS)
2m vs_1z	**100.0**(0.0)	**100.0**(0.0)	**100.0**(0.0)	**95.3**(5.2)	/	*100.0*(0.0)	*100.0*(0.0)
3m	**100.0**(0.0)	**100.0**(1.5)	**100.0**(0.0)	96.9(1.3)	/	*100.0*(0.0)	*100.0*(1.5)
2svs1sc	**100.0**(0.0)	**100.0**(0.0)	**100.0**(1.5)	96.9(2.9)	*100.0*(0.0)	*100.0*(0.0)	*100.0*(0.0)
2s3z	**100.0**(0.7)	**100.0**(1.5)	**100.0**(0.0)	95.3(2.5)	*100.0*(0.0)	96.9(1.5)	96.9(1.5)
3svs3z	**100.0**(0.0)	**100.0**(0.0)	**100.0**(0.0)	**96.9**(12.5)	/	*100.0*(0.0)	*100.0*(0.0)
3svs4z	**100.0**(1.3)	**98.4**(1.6)	**99.2**(1.5)	**97.7**(1.7)	/	*100.0*(2.1)	*100.0*(1.5)
so many baneling	**100.0**(0.0)	**100.0**(0.7)	**100.0**(1.5)	96.9(2.3)	/	*100.0*(1.5)	96.9(1.5)
8m	**100.0**(0.0)	**100.0**(0.0)	**100.0**(0.7)	97.7(1.9)	/	*100.0*(0.0)	*100.0*(0.0)
MMM	**96.9**(0.6)	93.8(1.5)	**96.9**(0.0)	**95.3**(2.5)	/	*93.8*(2.6)	*96.9*(1.5)
1c3s5z	**100.0**(0.0)	96.9(2.6)	**100.0**(0.0)	96.1(1.7)	*100.0*(0.0)	*100.0*(0.0)	96.9(2.6)
bane vs bane	**100.0**(0.0)	**100.0**(0.0)	**100.0**(0.0)	**100.0**(0.0)	*100.0*(46.4)	*100.0*(0.0)	*100.0*(0.0)
3svs5z	**100.0**(0.6)	**99.2**(1.4)	**100.0**(0.0)	**98.4**(2.4)	78.9(4.2)	*98.4*(5.5)	*100.0*(1.2)
2cvs64zg	**100.0**(0.0)	**100.0**(0.0)	98.4(1.3)	92.2(4.0)	*100.0*(0.0)	*96.9*(3.1)	95.3(3.5)
8mvs9m	**96.9**(0.6)	**96.9**(0.6)	**96.9**(0.7)	92.2(2.0)	/	*84.4*(5.1)	*87.5*(2.1)
25m	**100.0**(1.5)	**100.0**(4.0)	**100.0**(0.0)	85.9(7.1)	/	*96.9*(3.1)	*93.8*(2.9)
5mvs6m	**89.1**(2.5)	**88.3**(1.2)	**87.5**(2.3)	75.8(3.7)	*71.1*(9.2)	*65.6*(14.1)	*68.8*(8.2)
3s5z	**96.9**(0.7)	**96.9**(1.9)	**96.9**(1.5)	88.3(2.9)	*93.8*(2.0)	71.9(11.8)	53.1(15.4)
10mvs11m	**96.9**(4.8)	**96.9**(1.2)	**93.0**(7.4)	**95.3**(1.0)	*95.3*(2.2)	81.2(8.3)	*89.1*(5.5)
MMM2	**90.6**(2.8)	**87.5**(5.1)	**86.7**(7.3)	**87.5**(2.6)	*89.8*(6.7)	51.6(21.9)	28.1(29.6)
3s5zvs3s6z	**84.4**(34.0)	63.3(19.2)	**82.8**(19.1)	**82.8**(5.3)	*96.8*(25.11)	*75.0*(36.3)	18.8(37.4)
27mvs30m	**93.8**(2.4)	85.9(3.8)	69.5(11.8)	39.1(9.8)	*96.8*(1.5)	*93.8*(3.8)	*89.1*(6.5)
6hvs8z	**88.3**(3.7)	**85.9**(30.9)	**84.4**(33.3)	9.4(2.0)	*78.1*(37.0)	*78.1*(5.6)	*81.2*(31.8)
corridor	**100.0**(1.2)	**98.4**(0.8)	**98.4**(3.1)	84.4(2.5)	*65.6*(32.1)	*93.8*(3.5)	*93.8*(2.8)

Table 1: Median evaluation win rate and standard deviation on all the SMAC maps for different methods, Columns with "*" display results using the same number of timesteps as RODE. We bold all values within 1 standard deviation of the maximum and among the "*" columns, we denote all values within 1 standard deviation of the maximum with underlined italics.

Appendix E. MAPPO and QMix are trained until convergence or reaching 10M environment steps while results for other methods are directly calculated using the released training statistics.

We observe that IPPO and MAPPO achieve strong performance in the vast majority of SMAC maps. In particular, MAPPO (*AS* and *FP*) and IPPO perform at least as well as QMix in most maps despite using the same number of samples. Comparing different value functions inputs, we observe that the performance of IPPO and MAPPO(*AS*) are highly similar, with the methods performing strongly in all but 1 map each. MAPPO with the *FP* state performs well on all maps, achieving a median win-rate of least 84% in all scenarios. In comparison to RODE, we observe that PPO based methods, in particular MAPPO(FP), achieves performance comparable or superior to RODE's in 10 of 14 maps, while using the same number of training samples. With additional samples, the performance of PPO based methods continues to improve, and ultimately matches or exceeds RODE's performance in nearly every map. Overall, MAPPO's effectiveness in nearly every SMAC map suggests that simple PPO-based algorithms are strong baselines in many multi-agent problems.

4.3 Hanabi Results

emphWe evaluate MAPPO in the 2-player full-scale Hanabi game, trained without auxiliary tasks, and compare with the results of several strong off-policy methods, namely SAD [13], a Q-learning variant that has been successful in Hanabi, and the model-free SOTA achieved by a variant of Value Decomposition Networks (VDN) [31] which utilizes prioritized experience replay [27] and auxiliary tasks[1]. The result for VDN is obtained directly from [14]. The reported result for SAD does not use auxiliary training tasks, which have been shown to improve performance in 2-player Hanabi [13].

Algorithms	Best	Avg.
SAD(10B)	24.01	23.87
VDN(7B)	24.29	/
MAPPO(7.2B)	24.03	23.89
MAPPO(18.5B)	24.23	/

Table 2: Best and Average evaluation scores of MAPPO and SAD on Hanabi-Full with 2 players. Values in parenthesis indicate the number of timesteps used.

[1]The overall SOTA score in 2 player Hanabi is 24.52, achieved by a search-based algorithm [14].

Because each agent's local observation does not contain information about the agent's own cards[2], we implement a function which creates a global state by adding the agents' own cards to the local observation. SAD uses about 10B timesteps[3] and reports the best and average score over 13 trials; VDN is trained for approximately 7B timesteps and reports results over 1 seed. Due to computational constraints, we train MAPPO for 7.2B environment steps over 4 random seeds and 18.5B environment steps over 1 seed.

As demonstrated in Table 2, although it uses fewer training steps than SAD, MAPPO is able to produce results comparable to the best and average SAD rewards and even continues to improve with longer training. However, the VDN method achieves stronger results than MAPPO, possibly as a result of utilizing auxiliary tasks during training. Incorporating auxiliary tasks into MAPPO would be an interesting direction of future investigation.

5 Ablation Studies

In this section, we examine the aspects of MAPPO which are most critical to its performance by presenting ablation studies of the 5 suggestions described in Sec. 3: value normalization, value function input representation, training data usage, policy clipping, and death masking. Unless otherwise indicated, all MAPPO results in this section are obtained using the *FP* global state.

5.1 Value Normalization

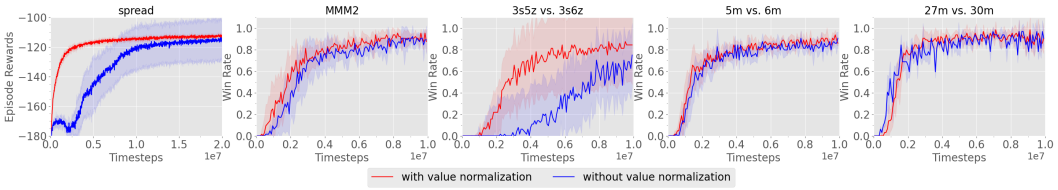


Figure 5: Impact of value normalization on MAPPO's performance in SMAC and MPE.

We evaluate MAPPO with and without the value normalization described in Sec. 3.1, and display the results in Fig 5. In *Spread*, where the episode rewards range from below -200 to 0, value normalization is critical to strong performance. Value normalization also has positive impacts on several SMAC maps, either by improving final performance or by reducing the training variance.

5.2 Input Representation to Value Function

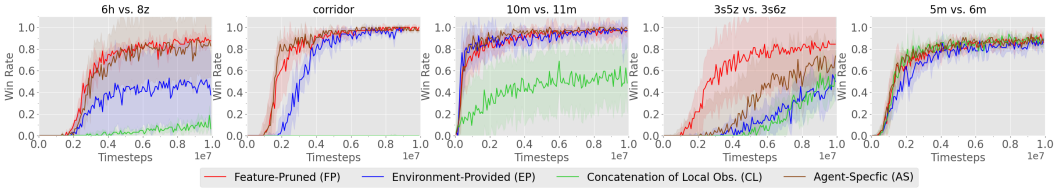


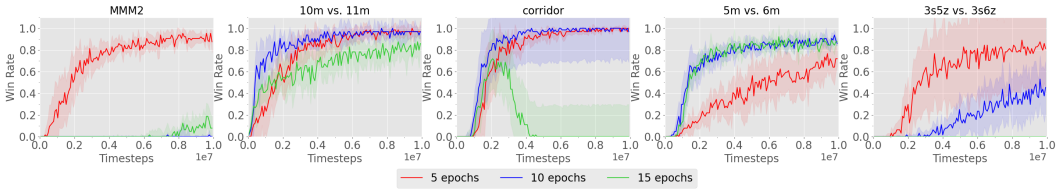
Figure 6: Effect of different value function input representations.

We compare each of the centralized value function input representations discussed in Sec. 3 in various SMAC maps. The results in Fig. 6 demonstrate that using the concatenation of local observations (*CL*), which is much higher dimensional than the other global states, is ineffective, particularly in maps with many agents. In comparison, using the environment-provided (*EP*) global state achieves stronger performance but notably achieves subpar performance in more difficult maps, likely due to the lack of important local information. The agent-specific (*AS*) and feature-pruned (*FP*) global states both achieve strong performance, with the *FP* state outperforming *AS* states on several maps. This demonstrates that state dimensionality, agent-specific features, and global information are all important in forming an effective global state.

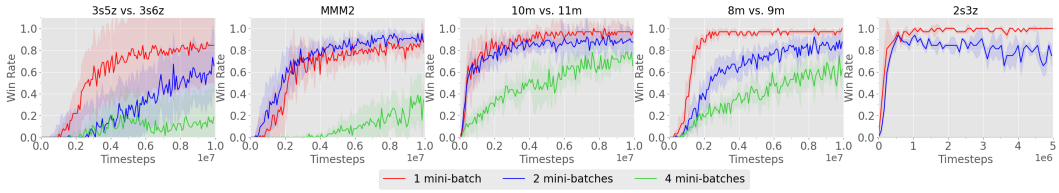
[2]The local observations in Hanabi contain information about the other agent's cards and game state.

[3]Reported in communication with authors

5.3 Training Data Usage



(a) effect of different training epochs.



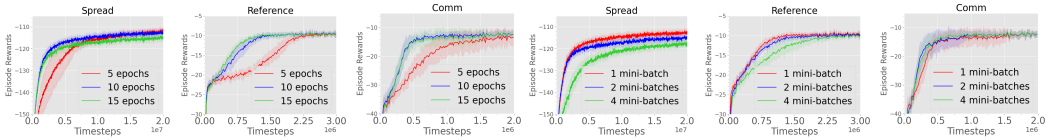
(b) effect of different mini-batch numbers.

Figure 7: The effect of epoch and mini-batch number on MAPPO's performance in SMAC.

We first examine the effect of training epochs in various SMAC maps in Fig. 7(a). We observe detrimental effects when training with large epoch numbers: when training with 15 epochs, MAPPO consistently learns a suboptimal policy, with particularly poor performance in the very difficult MMM2 and Corridor maps. In comparison, MAPPO performs well when using 5 or 10 epochs.

The performance of MAPPO is also highly sensitive to the number of mini-batches per training epoch. We consider three mini-batch values: 1, 2, and 4. A mini-batch of 4 indicates that we split the training data into 4 mini-batches to run gradient descent. Fig. 7(b) demonstrates that using more mini-batches negatively affects MAPPO's performance: when using 4 mini-batches, MAPPO fails to solve any of the selected maps while using 1 mini-batch produces the best performance on 22/23 maps.

As shown in Fig. 8, similar conclusions can be drawn in the MPE tasks. In *Reference* and *Comm*, the simplest MPE tasks, all chosen epoch and minibatch values result in the same final performance, and using 15 training epochs even leads to faster convergence. However, in the harder *Spread* task, we observe a similar trend to SMAC: fewer epochs and no mini-batch splitting produces the best results.



(a) effect of different training epochs.

(b) effect of different mini-batch numbers.

Figure 8: The effect of epoch and mini-batch number on MAPPO's performance in MPE.

5.4 PPO Clipping

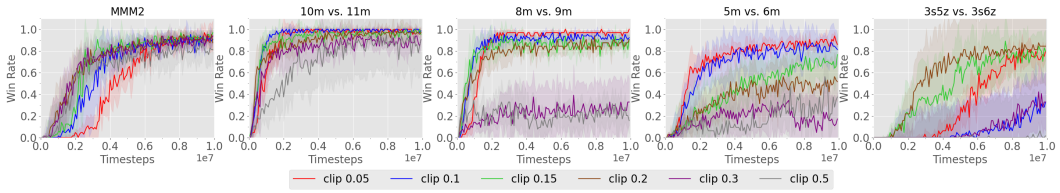


Figure 9: Impact of PPO clipping ratio on MAPPO's performance in SMAC.

We study the impact of PPO clipping strengths, controlled by the ϵ hyperparameter, in Fig. 9. Note that ϵ is the same for both policy and value clipping. We generally observe that MAPPO's performance is sensitive to the clipping strength. With small ϵ terms such as 0.05, MAPPO's learning speed is slowed in several maps, including MMM2 and 3s5z vs. 3s6z. However, with smaller ϵ terms, the performance is more stable, as demonstrated by the smaller standard deviation in the training curves.

We also observe that large ϵ terms such as 0.3 and 0.5, which allow for larger updates to the policy and value function per gradient step, can result in particularly poor performance.

5.5 Death Masking

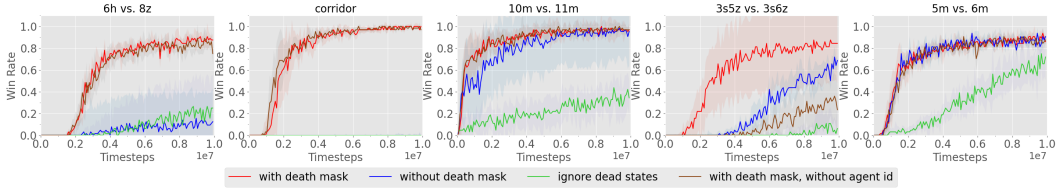


Figure 10: The effect of death mask on MAPPO's performance in SMAC.

We consider 4 death masking variants: (1) our suggested death masking, in which we replace the value state for a dead agent with a zero state containing the agent ID; (2) MAPPO without death masking, i.e., still using the nonzero global state as value input; (3) completely drop the transition samples after an agent dies (note that we still need to accumulate rewards after the agent dies to correctly estimate episode returns); and (4) replacing the global state with a pure zero-state which does not include the agent ID. Fig. 10 demonstrates that variant (1) significantly outperforms variants (2) and (3), and consistently achieves overall strong performance. Including the agent id in the death mask, as is done in variant (1), is particularly important in maps with heterogenous agents, as demonstrated by the superior performance of variant (1) compared to variant (4), which does not contain the agent ID in the death-mask zero-state, in the 3s5z vs. 3s6z map.

6 Conclusion

This work demonstrates that MAPPO, an on-policy policy gradient multi-agent reinforcement learning algorithm, achieves strong results comparable to the state-of-the-art on a variety of cooperative multi-agent challenges. Despite its on-policy nature, MAPPO is competitive to ubiquitous off-policy methods such as MADDPG, QMix, and RODE in terms of final performance, and in the vast majority of cases, comparable to off-policy methods in terms of sample-efficiency.

Additionally, in Sections 3 and 5, we demonstrate 5 key algorithmic and implementation techniques that are important to MAPPO's performance and support our findings with a variety of ablation studies which empirically demonstrate the effect of these techniques on MAPPO's performance.

There are several limitations to our work. Firstly, our benchmark environments all use discrete action spaces, are all cooperative, and in the vast majority of cases, contain homogeneous agents. In future work, we aim to test MAPPO on a wider range of domains such as competitive games and multi-agent problems with continuous action spaces and heterogeneous agents. Furthermore, our work is primarily empirical in nature, and does not directly analyze the theoretical underpinnings of MAPPO. We believe that the empirical analysis of our suggestions can serve as starting points for further analysis into MAPPO's properties.

Overall, the strong results obtained by MAPPO ultimately suggest that properly configured MAPPO is a competitive baseline for MARL tasks.

References

[1] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? a large-scale study. In *International Conference on Learning Representations*, 2021.

[2] Bowen Baker, Ingmar Kanitscheider, Todor M. Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[3] Nolan Bard, Jakob N Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, et al. The Hanabi challenge: A new frontier for AI research. *Artificial Intelligence*, 280:103216, 2020.

[4] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[5] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998(746-752):2, 1998.

[6] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip H. S. Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[7] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338, 2016.

[8] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International Conference on Learning Representations*, 2020.

[9] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416, 2018.

[10] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[11] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[13] Hengyuan Hu and Jakob N Foerster. Simplified action decoder for deep multi-agent reinforcement learning. In *International Conference on Learning Representations*, 2020.

[14] Hengyuan Hu, Adam Lerer, Noam Brown, and Jakob Nicolaus Foerster. Learned belief search: Efficiently improving policies in partially observable settings, 2021.

[15] Andrew Ilyas, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. A closer look at deep policy gradients. In *International Conference on Learning Representations*, 2020.

[16] Shariq Iqbal, Christian A. Schröder de Witt, Bei Peng, Wendelin Böhmer, Shimon Whiteson, and Fei Sha. Ai-qmix: Attention and imagination for dynamic multi-agent reinforcement learning. *CoRR*, abs/2006.04222, 2020.

[17] Bobby Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does sgd escape local minima? In *International Conference on Machine Learning*, pages 2698–2707. PMLR, 2018.

[18] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[19] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*, 2017.

[20] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

[21] Frans A Oliehoek, Christopher Amato, et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

[22] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

[23] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Comparative evaluation of multi-agent deep reinforcement learning algorithms. *arXiv preprint arXiv:2006.07869*, 2020.

[24] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. In *NeurIPS*, 2020.

[25] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4295–4304. PMLR, 10–15 Jul 2018.

[26] Mikayel Samvelyan, Tabish Rashid, Christian Schröder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob N. Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *CoRR*, abs/1902.04043, 2019.

[27] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016.

[28] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[30] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5887–5896. PMLR, 2019.

[31] 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 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2085–2087, 2018.

[32] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[33] George Tucker, Surya Bhupatiraju, Shixiang Gu, Richard Turner, Zoubin Ghahramani, and Sergey Levine. The mirage of action-dependent baselines in reinforcement learning. In *International conference on machine learning*, pages 5015–5024. PMLR, 2018.

[34] Oriol Vinyals, Igor Babuschkin, M Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, H David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, P John Agapiou, Max Jaderberg, S Alexander Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin

Dalibard, David Budden, Yury Sulsky, James Molloy, L Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

[35] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. {QPLEX}: Duplex dueling multi-agent q-learning. In *International Conference on Learning Representations*, 2021.

[36] Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. RODE: Learning roles to decompose multi-agent tasks. In *International Conference on Learning Representations*, 2021.

[37] Yihan Wang, Beining Han, Tonghan Wang, Heng Dong, and Chongjie Zhang. {DOP}: Off-policy multi-agent decomposed policy gradients. In *International Conference on Learning Representations*, 2021.

A MAPPO Details

Algorithm 1 Recurrent-MAPPO

Initialize θ, the parameters for policy π and ϕ, the parameters for critic V, using Orthogonal initialization (Hu et al., 2020)
Set learning rate α
while $step \leq step_{\text{max}}$ **do**
  set data buffer $D = \{\}$
  for $i = 1$ **to** $batch\_size$ **do**
    $\tau = []$ empty list
    initialize $h_{0,\pi}^{(1)}, \dots h_{0,\pi}^{(n)}$ actor RNN states
    initialize $h_{0,V}^{(1)}, \dots h_{0,V}^{(n)}$ critic RNN states
    for $t = 1$ **to** T **do**
      for all agents a **do**
        $p_t^{(a)}, h_{t,\pi}^{(a)} = \pi(o_t^{(a)}, h_{t-1,\pi}^{(a)}; \theta)$
        $u_t^{(a)} \sim p_t^{(a)}$
        $v_t^{(a)}, h_{t,V}^{(a)} = V(s_t^{(a)}, h_{t-1,V}^{(a)}; \phi)$
      end for
      Execute actions $\boldsymbol{u_t}$, observe $r_t, s_{t+1}, \boldsymbol{o_{t+1}}$
      $\tau \mathrel{+}= [s_t, \boldsymbol{o_t}, \boldsymbol{h_{t,\pi}}, \boldsymbol{h_{t,V}}, \boldsymbol{u_t}, r_t, s_{t+1}, \boldsymbol{o_{t+1}}]$
    end for
    Compute advantage estimate $\hat{A}$ via GAE on τ, using PopArt
    Compute reward-to-go $\hat{R}$ on τ and normalize with PopArt
    Split trajectory τ into chunks of length L
    for l = 0, 1, .., T//L **do**
      $D = D \cup (\tau[l : l + T, \hat{A}[l : l + L], \hat{R}[l : l + L])$
    end for
  end for
  for mini-batch $k = 1, \dots, K$ **do**
    $b \leftarrow$ random mini-batch from D with all agent data
    for each data chunk c in the mini-batch b **do**
      update RNN hidden states for π and V from first hidden state in data chunk
    end for
  end for
  Adam update θ on $L(\theta)$ with data b
  Adam update ϕ on $L(\phi)$ with data b
end while

MAPPO trains two separate neural networks: an actor network with parameters θ, and a value function network (referred to as a critic) with parameters ϕ. These networks can be shared amongst all agents if the agents are homogeneous, but each agent can also have its own pair of actor and critic networks. We assume here that all agents share critic and actor networks, for notational convenience. Specifically, the critic network, denoted as V_ϕ, performs the following mapping: $S \rightarrow \mathbb{R}$. The global state can be agent-specific or agent-agnostic.

The actor network, denoted as π_θ, maps agent observations $o_t^{(a)}$ to a categorical distribution over actions in discrete action spaces, or to the mean and standard deviation vectors of a Multivariate Gaussian Distribution, from which an action is sampled, in continuous action spaces.

The actor network is trained to maximize the objective
$L(\theta) = [\frac{1}{Bn} \sum_{i=1}^{B} \sum_{k=1}^{n} \min(r_{\theta,i}^{(k)} A_i^{(k)}, \text{clip}(r_{\theta,i}^{(k)}, 1-\epsilon, 1+\epsilon) A_i^{(k)})] + \sigma \frac{1}{Bn} \sum_{i=1}^{B} \sum_{k=1}^{n} S[\pi_\theta(o_i^{(k)}))]$, where $r_{\theta,i}^{(k)} = \frac{\pi_\theta(a_i^{(k)}|o_i^{(k)})}{\pi_{\theta_{old}}(a_i^{(k)}|o_i^{(k)})}$. $A_i^{(k)}$ is computed using the GAE method, S is the policy entropy, and σ is the entropy coefficient hyperparameter.

The critic network is trained to minimize the loss function
$L(\phi) = \frac{1}{Bn} \sum_{i=1}^{B} \sum_{k=1}^{n} (\max[(V_\phi(s_i^{(k)}) - \hat{R}_i)^2, (\text{clip}(V_\phi(s_i^{(k)}), V_{\phi_{old}}(s_i^{(k)}) - \varepsilon, V_{\phi_{old}}(s_i^{(k)}) + \varepsilon) - \hat{R}_i)^2],$
where $\hat{R}_i$ is the discounted reward-to-go.

In the loss functions above, B refers to the batch size and n refers to the number of agents.

If the critic and actor networks are RNNs, then the loss functions additionally sum over time, and the networks are trained via Backpropagation Through Time (BPTT). Pseudocode for recurrent-MAPPO is shown in Alg. 1.

B Justification of Death Masking

Let $\mathbf{0}_a$ be a zero vector with agent a's agent ID appended to the end. The use of agent ID leads to an agent-specific value function depending on an agent's type or role. It has been empirically justified in Sec. 6.5 that such an agent-specific feature is particularly helpful when the environment contains heterogeneous agents.

We now provide some intuition as to why using $\mathbf{0_a}$ as the critic input when agents are dead appears to be a better alternative to using the usual agent-specific global state as the input to the value function. Note that our global state to the value network has agent-specific information, such as available actions and relative distances to other agents. When an agent dies, these agent-specific features become zero, while the remaining agent-agnostic features remain nonzero - this leads to a drastic distribution shift in the critic input compared to states in which the agent is alive. In most SMAC maps, an agent is dead in only a small fraction of the timesteps in a batch (about 20%); due to their relative infrequency in the training data the states in which an agent is dead will likely have large value prediction error. Moreover, it is also possible that training on these out of distribution inputs harms the feature representation of the value network.

Although replacing the states at which an agent is dead with a fixed vector $\mathbf{0_a}$ also results in a distribution shift, the replacement results in there being only 1 vector which captures the state at which an agent is dead - thus, the critic is more likely to be able to fit the average post-death reward for agent a to the input $\mathbf{0_a}$. Our ablation on the value function fitting error provide some weight to this hypothesis.

Another possible mechanism of handling agent deaths is to completely skip value learning in states in which an agent is dead, by essentially terminating an agent's episode when it dies. Suppose the game episode is T and the agent dies at timestep d. If we are not learning on dead state then, in order to correctly accumulate the episode return, we need to replace the reward r_d at timestep d by the total return R_d at time d, i.e., $r_d \leftarrow R_d = \sum_{t=d}^{T} \gamma^{t-d} r_t$. We would then need to compute the GAE only on those states in which the agent is alive. While this approach is theoretically correct (we are simply treating the state where the agent died as a terminal state and assigning the accumulated discounted reward as a terminal reward), it can have negative ramifications in the policy learning process, as outlined below.

The GAE is an exponentially weighted average of k-step returns intended to trade off between bias and variance. Large k values result in a low bias, but high variance return estimate, whereas small k values result in a high bias, low variance return estimate. However, since the entire post death return R_d replaces the single timestep reward r_d at timestep d, computing the 1-step return estimate at timestep d essentially becomes a $(T - d)$-step estimate, eliminating potential benefits of value function truncation of the trajectory and potentially leading to higher variance. This potentially dampens the benefit that could come from using the GAE at the timesteps in which an agent is dead.

We analyze the impact of the death masking by comparing different ways of handling dead agents, including: (1) our death masking, (2) using global states without death masking and (3) ignoring dead states in value learning and in the GAE computation. We first examine the median win rate with these different options in Fig. 20 and 22. It is evident that our method of death masking, which uses $\mathbf{0_a}$ as the input to the critic when an agent is dead, results in superior performance compared to other options.

Additionally, Fig. 23 demonstrates that using the death mask results in a lower values loss in the vast majority of SMAC maps, demonstrating that the accuracy of the value predictions improve when

using the death mask. While the arguments here are intuitive the clear experimental benefits suggest that theoretically characterizing the effect of this method would be valuable.

C Testing domains

Multi-agent Particle-World Environment (MPE) was introduced in (Lowe et al., 2017). MPE consist of various multi-agent games in a 2D world with small particles navigating within a square box. We consider the 3 fully cooperative tasks from the original set shown in Fig. 1(a): *Spread*, *Comm*, and *Reference*. Note that since the two agents in *speaker-listener* have different observation and action spaces, this is the only setting in this paper where we do not share parameters but train separate policies for each agent.

StarCraftII Micromanagement Challenge (SMAC) tasks were introduced in (Rashid et al., 2019). In these tasks, decentralized agents must cooperate to defeat adversarial bots in various scenarios with a wide range of agent numbers (from 2 to 27). We use a global game state to train our centralized critics or Q-functions. Fig. 1(c) and 1(d) show two example StarCraftII environments.

As described in Sec. 5.2, we utilize an agent-specific global state as input to the global state. This agent-specific global state augments the original global state provided by the SMAC environment by adding relevant agent-specific features.

Specifically, the original global state of SMAC contains information about all agents and enemies - this includes information such as the distance from each agent/enemy to the map center, the health of each agent/enemy, the shield status of each agent/enemy, and the weapon cooldown state of each agent. However, when compared to the local observation of each agent, the global state does not contain agent-specific information including agent id, agent movement options, agent attack options, relative distance to allies/enemies. Note that the local observation contains information only about allies/enemies within a sight radius of the agent. To address the lack of critical local information in the environment provided global state, we create several other global inputs which are specific to each agent, and combine local and global features. The first, which we call *agent-specific (AS)*, uses the concatenation of the environment provided global state and agent *i*'s observation, o_i, as the global input to MAPPO's critic during gradient updates for agent *i*. However, since the global state and local agent observations have overlapping features, we additionally create a feature-pruned global state (*FP*) which removes the overlapping features in the *AS* global state.

Hanabi is a turn-based card game, introduced as a MARL challenge in (Bard et al., 2020) , where each agent observes other players' cards except their own cards. A visualization of the game is shown in Fig. 1(b). The goal of the game is to send information tokens to others and cooperatively take actions to stack as many cards as possible in ascending order to collect points.

The turn-based nature of Hanabi presents a challenge when computing the reward for an agent during it's turn. We utilize the forward accumulated reward as one turn reward R_i; specifically, if there are 4 players and players 0, 1, 2, and 3 execute their respective actions at timesteps *k, k+1, k+2, k+3* respectively, resulting in rewards of $r_k^{(0)}, r_{k+1}^{(1)}, r_{k+2}^{(2)}, r_{k+3}^{(3)}$, then the reward assigned to player 0 will be $R_0 = r_k^{(0)} + r_{k+1}^{(1)} + r_{k+2}^{(2)} + r_{k+3}^{(3)}$ and similarly, the reward assigned to player 1 will be $R_1 = r_{k+1}^{(1)} + r_{k+2}^{(2)} + r_{k+3}^{(3)} + r_{k+4}^{(0)}$. Here, r_t^i denotes the reward received at timestep t when agent i is executes a move.

D Training details

D.1 Implementation

All algorithms utilize parameter sharing - i.e., all agents share the same networks - in all environments except for the *Comm* scenario in the MPE. Furthermore, we tune the architecture and hyperparameters of MADDPG and QMix, and thus use different hyperparameters than the original implementations. However, we ensure that the performance of the algorithms in the baselines matches or exceeds the results reported in their original papers.

For each algorithm, certain hyperparameters are kept constant across all environments; these are listed in Tables 5 and 6 for MAPPO, QMix, and MADDPG, respectively. These values are obtained either

from the PPO baselines implementation in the case of MAPPO, or from the original implementations for QMix and MADDPG. Note that since we use parameter sharing and combine all agents' data, the actual batch-sizes will be larger with more agents.

In these tables, "recurrent data chunk length" refers to the length of chunks that a trajectory is split into before being used for training via BPTT (only applicable for RNN policies). "Max clipped value loss" refers to the value-clipping term in the value loss. "Gamma" refers to the discount factor, and "huber delta" specifies the delta parameter in the Huber loss function. "Epsilon" describes the starting and ending value of ϵ for ϵ-greedy exploration, and "epsilon anneal time" refers to the number of environment steps over which ϵ will be annealed from the starting to the ending value, in a linear manner. "Use feature normalization" refers to whether the feature normalization is applied to the network input.

D.2 Hyperparameters

Tables 3-11 describe the common hyperparameters, hyperparameter grid search values, and chosen hyperparmeters for MAPPO, QMix, and MADDPG in all testing domains. Tables 7, 8, and 9 describe common hyperparameters for different algorithms in each domain. Tables 10, 11, and 12 describe the hyperparameter grid search procedure for the MAPPO, QMix, and MADDPG algorithms, respectively. Lastly, Tables 13, 14, and 15 describe the final chosen hyperparameters among fine-tuned parameters for different algorithms in MPE, SMAC, and Hanabi, respectively.

For MAPPO, "Batch Size" refers to the number of environment steps collected before updating the policy via gradient descent. Since agents do not share a policy only in the MPE speaker-listener, the batch size does not depend on the number of agents in the speaker-listener environment. "Mini-batch" refers to the number of mini-batches a batch of data is split into, "gain" refers to the weight initialization gain of the last network layer for the actor network. "Entropy coef" is the entropy coefficient σ in the policy loss. "Tau" corresponds to the rate of the polyak average technique used to update the target networks, and if the target networks are not updated in a "soft" manner, the "hard interval" hyperparameter specifies the number of gradient updates which must elapse before the target network parameters are updated to equal the live network parameters. "Clip" refers to the ϵ hyperparameter in the policy objective and value loss which controls the extent to which large policy and value function changes are penalized.

MLP network architectures are as follows: all MLP networks use "num fc" linear layers, whose dimensions are specified by the "fc layer dim" hyperparameter. When using MLP networks, "stacked frames" refers to the number of previous observations which are concatenated to form the network input: for instance, if "stacked frames" equals 1, then only the current observation is used as input, and if "stacked frames" is 2, then the current and previous observations are concatenated to form the input. For RNN networks, the network architecture is "num fc" fully connected linear layers of dimension "fc layer dim", followed by "num GRU layers" GRU layers, finally followed by "num fc after" linear layers.

E Additional Results

E.1 Additional Hanabi Results

Fig. 11 shows the score of MAPPO in 2-player Hanabi over 4 seeds for 7.2 B environment steps. Fig. 12 shows the score of MAPPO in 2-player Hanabi over 4 seeds for 18.5 B environment steps.

E.2 Additional SMAC Results

Results of all algorithms in all SMAC maps can be found in Tab. 3 and 4.

As MAPPO does not converge within 10M environment steps in the 3s5z vs. 3s6z map, Fig. 13 shows the performance of MAPPO in 3s5z vs. 3s6z when run until convergence. Fig. 14 presents the evaluation win of MAPPO with different value inputs (*FP* and *AS*), decentralized PPO (IPPO), QMix, and QMix with a modified global state input to the mixer network, which we call QMix (MG). Specifically, QMix(MG) uses a concatenation of the default environment global state, as well as *all* agents' local observations, as the mixer network input.

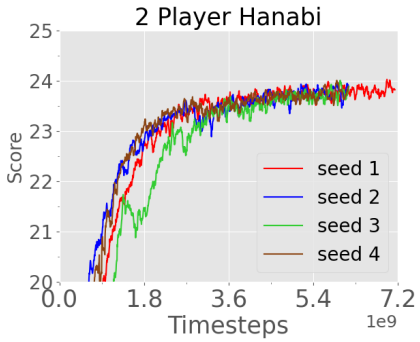


Figure 11: Score of MAPPO in 2-player Hanabi at 7.2 B environment steps.

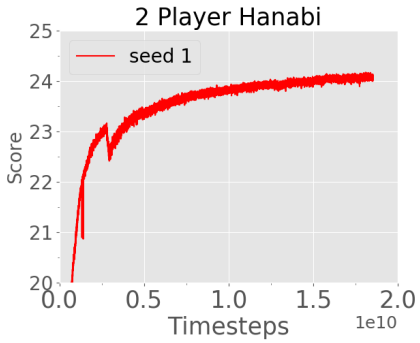


Figure 12: Score of MAPPO in 2-player Hanabi at 18.5 B environment steps.

Fig. 15 compares the results of MAPPO(FP) to various off-policy baselines, including QMix(MG), RODE, QPLEX, CWQMix, and AIQMix, in many SMAC maps.

F Ablation Studies

We present the learning curves for all ablation studies performed. Fig. 16 demonstrates the impact of value normalization on MAPPO's performance. Fig. 17 shows the effect of global state information on MAPPO's performance in SMAC. Fig. 18 studies the influence of training epochs on MAPPO's performance. Fig. 19 studies the influence of clipping term on MAPPO's performance. Fig. 20 and Fig. 21 illustrates the influence of the death mask on MAPPO(FP)'s and MAPPO(AS)'s performance. Similarly, Fig. 22 compares the performance of MAPPO when ignoring states in which an agent is dead when computing GAE to using the death mask when computing the GAE. Fig. 23 illustrates the effect of death mask on MAPPO's value loss in the SMAC domain. Lastly, Fig. 24 shows the influence of including the agent-id in the agent-specific global state.

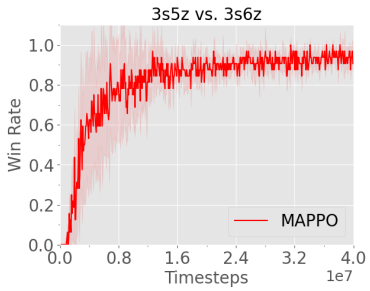


Figure 13: Median win rate of 3s5z vs. 3s6z map after 40M environment steps.

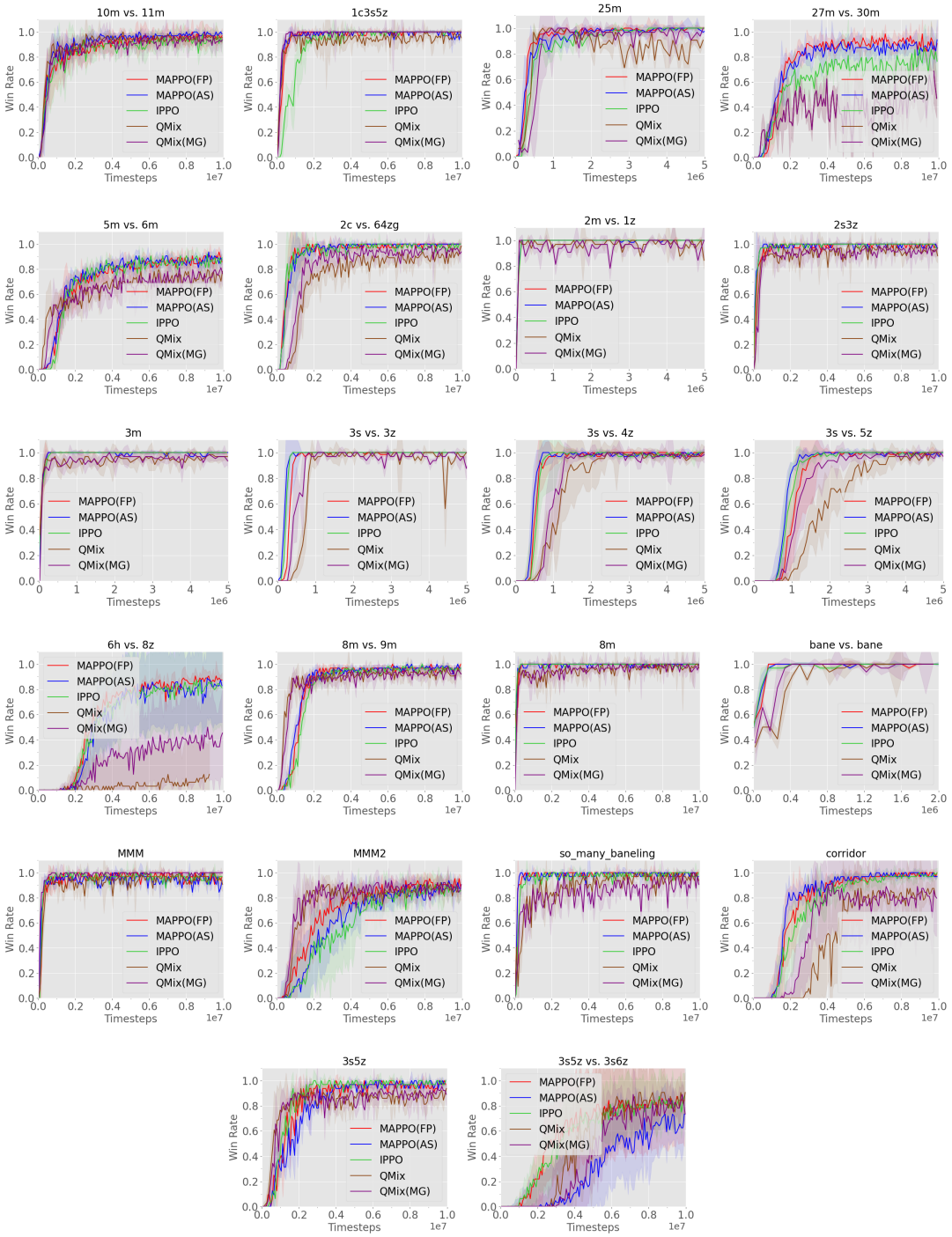


Figure 14: Median evaluation win rate of 23 maps in the SMAC domain.

Map	Map Difficulty	MAPPO(FP)	MAPPO(AS)	IPPO	QMix	QMix(MG)	RODE	QPLEX	CWQMix	AIQMix
2m_vs_1z	Easy	100.0(0.0)	100.0(0.0)	100.0(0.0)	95.3(5.2)	96.9(4.5)	/	/	/	/
3m	Easy	100.0(0.0)	100.0(1.5)	100.0(0.0)	96.9(1.3)	96.9(1.7)	/	/	/	/
2s_vs_1sc	Easy	100.0(0.0)	100.0(0.0)	100.0(1.5)	96.9(2.9)	100.0(1.4)	100(0.0)	98.4(1.6)	100(0.0)	100(0.0)
2s3z	Easy	100.0(0.7)	100.0(1.5)	100.0(0.0)	95.3(2.5)	96.1(2.1)	100(0.0)	100(4.3)	93.7(2.2)	96.9(0.7)
3s_vs_3z	Easy	100.0(0.0)	100.0(0.0)	100.0(0.0)	96.9(12.5)	96.9(3.7)	/	/	/	/
3s_vs_4z	Easy	100.0(1.3)	98.4(1.6)	99.2(1.5)	97.7(1.9)	97.7(1.4)	/	/	/	/
so_many_baneling	Easy	100.0(0.0)	100.0(0.7)	100.0(1.5)	96.9(2.3)	92.2(5.8)	/	/	/	/
8m	Easy	100.0(0.0)	100.0(0.0)	100.0(0.7)	97.7(1.9)	96.9(2.0)	/	/	/	/
MMM	Easy	96.9(2.6)	93.8(1.5)	96.9(0.0)	95.3(2.5)	100.0(0.0)	/	/	/	/
1c3s5z	Easy	100.0(0.0)	96.9(2.6)	100.0(0.0)	96.1(1.7)	100.0(0.5)	100(0.0)	96.8(1.6)	96.9(1.4)	92.2(10.4)
bane_vs_bane	Easy	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.9)	100.0(2.1)	100(46.4)	100(2.9)	100(0.0)	85.9(34.7)
3s_vs_5z	Hard	100.0(0.6)	99.2(1.4)	100.0(0.0)	98.4(2.4)	98.4(1.6)	78.9(4.2)	98.4(1.4)	34.4(6.5)	82.8(10.6)
2c_vs_64zg	Hard	100.0(0.0)	100.0(0.0)	98.4(1.3)	92.2(4.0)	95.3(1.5)	100(0.0)	90.6(7.3)	85.9(3.3)	97.6(2.3)
8m_vs_9m	Hard	96.9(0.6)	96.9(0.6)	96.9(0.7)	92.2(2.0)	93.8(2.7)	/	/	/	/
25m	Hard	100.0(1.5)	100.0(4.0)	100.0(0.0)	85.9(7.1)	96.9(3.8)	/	/	/	/
5m_vs_6m	Hard	89.1(2.5)	88.3(1.2)	87.5(2.3)	75.8(3.7)	76.6(2.6)	71.1(9.2)	70.3(3.2)	57.8(9.1)	64.1(5.5)
3s5z	Hard	96.9(0.7)	96.9(1.9)	96.9(1.5)	88.3(2.9)	92.2(1.8)	93.75(1.95)	96.8(2.2)	70.3(20.3)	96.9(2.9)
10m_vs_11m	Hard	96.9(4.8)	96.9(1.2)	93.0(7.4)	95.3(1.0)	92.2(2.0)	95.3(2.2)	96.1(8.7)	75.0(3.3)	96.9(1.4)
MMM2	Super Hard	90.6(2.8)	87.5(5.1)	86.7(7.3)	87.5(2.6)	88.3(2.2)	89.8(6.7)	82.8(20.8)	0.0(0.0)	67.2(12.4)
3s5z_vs_3s6z	Super Hard	84.4(34.0)	63.3(19.2)	82.8(19.1)	82.8(5.3)	82.0(4.4)	96.8(25.11)	10.2(11.0)	53.1(12.9)	0.0(0.0)
27m_vs_30m	Super Hard	93.8(2.4)	85.9(3.8)	69.5(11.8)	39.1(9.8)	39.1(9.8)	96.8(1.5)	43.7(18.7)	82.8(7.8)	62.5(34.3)
6h_vs_8z	Super Hard	88.3(3.7)	85.9(30.9)	84.4(33.3)	9.4(2.0)	39.8(4.0)	78.1(37.0)	1.5(31.0)	49.2(14.8)	0.0(0.0)
corridor	Super Hard	100.0(1.2)	98.4(0.8)	98.4(3.1)	84.4(2.5)	81.2(5.9)	65.6(32.1)	0.0(0.0)	0.0(0.0)	12.5(7.6)

Table 3: Median evaluation win rate and standard deviation on all the SMAC maps for different methods, using at most 10M training timesteps.

Map	Map Difficulty	MAPPO(FP)*	MAPPO(AS)*	IPPO*	QMix*	QMix(MG)*	RODE	QPLEX	CWQMix	AIQMix
2m_vs_1z	Easy	100.0(0.0)	100.0(0.0)	100.0(0.0)	96.9(2.8)	96.9(4.7)	/	/	/	/
3m	Easy	100.0(0.0)	100.0(1.5)	100.0(0.0)	92.2(2.7)	96.9(2.1)	/	/	/	/
2s_vs_1sc	Easy	100.0(0.0)	100.0(0.0)	100.0(0.0)	96.9(1.2)	96.9(4.6)	100(0.0)	98.4(1.6)	100(0.0)	100(0.0)
2s3z	Easy	96.9(1.5)	96.9(1.5)	100.0(0.0)	95.3(3.9)	92.2(2.3)	100(0.0)	100(4.3)	93.7(2.2)	96.9(0.7)
3s_vs_3z	Easy	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(1.5)	100.0(1.5)	/	/	/	/
3s_vs_4z	Easy	100.0(2.1)	100.0(1.5)	100.0(1.4)	87.5(3.2)	98.4(0.8)	/	/	/	/
so_many_baneling	Easy	100.0(1.5)	96.9(1.5)	96.9(1.5)	81.2(7.2)	78.1(6.7)	/	/	/	/
8m	Easy	100.0(0.0)	100.0(0.0)	100.0(1.5)	93.8(5.1)	93.8(2.7)	/	/	/	/
MMM	Easy	93.8(2.6)	96.9(1.5)	96.9(1.5)	95.3(3.9)	100.0(1.2)	/	/	/	/
1c3s5z	Easy	100.0(0.0)	96.9(2.6)	93.8(5.1)	95.3(1.2)	98.4(1.4)	100(0.0)	96.8(1.6)	96.9(1.4)	92.2(10.4)
bane_vs_bane	Easy	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100(46.4)	100(2.9)	100(0.0)	85.9(34.7)
3s_vs_5z	Hard	98.4(5.5)	100.0(1.2)	100.0(2.4)	56.2(8.8)	90.6(2.2)	78.9(4.2)	98.4(1.4)	34.4(6.5)	82.8(10.6)
2c_vs_64zg	Hard	96.9(3.1)	95.3(3.5)	93.8(9.2)	70.3(3.8)	84.4(3.7)	100(0.0)	90.6(7.3)	85.9(3.3)	97.6(2.3)
8m_vs_9m	Hard	84.4(5.1)	87.5(2.1)	76.6(5.6)	85.9(2.9)	85.9(4.7)	/	/	/	/
25m	Hard	96.9(3.1)	93.8(2.9)	93.8(5.0)	96.9(4.0)	93.8(5.7)	/	/	/	/
5m_vs_6m	Hard	65.6(14.1)	68.8(8.2)	64.1(7.7)	54.7(3.5)	56.2(2.1)	71.1(9.2)	70.3(3.2)	57.8(9.1)	64.1(5.5)
3s5z	Hard	71.9(11.8)	53.1(15.4)	84.4(12.1)	85.9(4.6)	89.1(2.6)	93.75(1.95)	96.8(2.2)	70.3(20.3)	96.9(2.9)
10m_vs_11m	Hard	81.2(8.3)	89.1(5.5)	87.5(17.5)	82.8(4.1)	85.9(2.3)	95.3(2.2)	96.1(8.7)	75.0(3.3)	96.9(1.4)
MMM2	Super Hard	51.6(21.9)	28.1(29.6)	26.6(27.8)	82.8(4.0)	79.7(3.4)	89.8(6.7)	82.8(20.8)	0.0(0.0)	67.2(12.4)
3s5z_vs_3s6z	Super Hard	75.0(36.3)	18.8(37.4)	65.6(25.9)	56.2(11.3)	39.1(4.7)	96.8(25.11)	10.2(11.0)	53.1(12.9)	0.0(0.0)
27m_vs_30m	Super Hard	93.8(3.8)	89.1(6.5)	73.4(11.5)	34.4(5.4)	34.4(5.4)	96.8(1.5)	43.7(18.7)	82.8(7.8)	62.5(34.3)
6h_vs_8z	Super Hard	78.1(5.6)	81.2(31.8)	78.1(33.1)	3.1(1.5)	29.7(6.3)	78.1(37.0)	1.5(31.0)	49.2(14.8)	0.0(0.0)
corridor	Super Hard	93.8(3.5)	93.8(2.8)	89.1(9.1)	64.1(14.3)	81.2(1.5)	65.6(32.1)	0.0(0.0)	0.0(0.0)	12.5(7.6)

Table 4: Median evaluation win rate and standard deviation on all the SMAC maps for different methods, Columns with "*" display results using the same number of timesteps as RODE.

common hyperparameters	value
recurrent data chunk length	10
gradient clip norm	10.0
gae lamda	0.95
gamma	0.99
value loss	huber loss
huber delta	10.0
batch size	num envs $\times$ buffer length $\times$ num agents
mini batch size	batch size / mini-batch
optimizer	Adam
optimizer epsilon	1e-5
weight decay	0
network initialization	Orthogonal
use reward normalization	True
use feature normalization	True

Table 5: Common hyperparameters used in MAPPO across all domains.

common hyperparameters	value
gradient clip norm	10.0
random episodes	5
epsilon	$1.0 \rightarrow 0.05$
epsilon anneal time	50000 timesteps
train interval	1 episode
gamma	0.99
critic loss	mse loss
buffer size	5000 episodes
batch size	32 episodes
optimizer	Adam
optimizer eps	1e-5
weight decay	0
network initialization	Orthogonal
use reward normalization	True
use feature normalization	True

Table 6: Common hyperparameters used in QMix and MADDPG across all domains.

hyperparameters	value
num envs	MAPPO: 128
buffer length	MAPPO: 25
num GRU layers	1
RNN hidden state dim	64
fc layer dim	64
num fc	2
num fc after	1

Table 7: Common hyperparameters used in the MPE domain for MAPPO, MADDPG, and QMix.

hyperparameters	value
num envs	MAPPO:8
buffer length	MAPPO: 400
num GRU layers	1
RNN hidden state dim	64
fc layer dim	64
num fc	2
num fc after	1

Table 8: Common hyperparameters used in the SMAC domain for MAPPO and QMix.

hyperparameters	value
num envs	1000
buffer length	100
fc layer dim	512
num fc	2

Table 9: Common hyperparameters used in the Hanabi domain for MAPPO.

Domains	lr	epoch	mini-batch	activation	clip	gain	entropy coef	network
MPE	[1e-4,5e-4,7e-4,1e-3]	[5,10,15,20]	[1,2,4]	[ReLU,Tanh]	[0.05,0.1,0.15,0.2,0.3,0.5]	[0.01,1]	/	[mlp,rnn]
SMAC	[1e-4,5e-4,7e-4,1e-3]	[5,10,15]	[1,2,4]	[ReLU,Tanh]	[0.05,0.1,0.15,0.2,0.3,0.5]	[0.01,1]	/	[mlp,rnn]
Hanabi	[1e-4,5e-4,7e-4,1e-3]	[5,10,15]	[1,2,4]	[ReLU,Tanh]	[0.05,0.1,0.15,0.2,0.3,0.5]	[0.01,1]	[0.01, 0.015]	[mlp,rnn]

Table 10: Sweeping procedure of MAPPO cross all domains.

Domains	lr	tau	hard interval	activation	gain
MPE	[1e-4,5e-4,7e-4,1e-3]	[0.001,0.005,0.01]	[100,200,500]	[ReLU,Tanh]	[0.01,1]
SMAC	[1e-4,5e-4,7e-4,1e-3]	[0.001,0.005,0.01]	[100,200,500]	[ReLU,Tanh]	[0.01,1]

Table 11: Sweeping procedure of QMix in the MPE and SMAC domains.

Domains	lr	tau	activation	gain	network
MPE	[1e-4,5e-4,7e-4,1e-3]	[0.001,0.005,0.01]	[ReLU,Tanh]	[0.01,1]	[mlp,rnn]

Table 12: Sweeping procedure of MADDPG in the MPE domain.

Scenarios	lr	gain	network	MAPPO			MADDPG		QMix		
				epoch	mini-batch	activation	tau	activation	tau	hard interval	activation
Spread	7e-4	0.01	rnn	10	1	Tanh	0.005	ReLU	/	100	ReLU
Reference	7e-4	0.01	rnn	15	1	ReLU	0.005	ReLU	0.005	/	ReLU
Comm	7e-4	0.01	rnn	15	1	Tanh	0.005	ReLU	0.005	/	ReLU

Table 13: Adopted hyperparameters used for MAPPO, MADDPG and QMix in the MPE domain.

Maps	lr	activation	MAPPO						QMix	
			epoch	mini-batch	clip	gain	network	stacked frames	hard interval	gain
2m vs. 1z	5e-4	ReLU	15	1	0.2	0.01	rnn	1	200	0.01
3m	5e-4	ReLU	15	1	0.2	0.01	rnn	1	200	0.01
2s vs. 1sc	5e-4	ReLU	15	1	0.2	0.01	rnn	1	200	0.01
3s vs. 3z	5e-4	ReLU	15	1	0.2	0.01	rnn	1	200	0.01
3s vs. 4z	5e-4	ReLU	15	1	0.2	0.01	mlp	4	200	0.01
3s vs. 5z	5e-4	ReLU	15	1	0.05	0.01	mlp	4	200	0.01
2c vs. 64zg	5e-4	ReLU	5	1	0.2	0.01	rnn	1	200	0.01
so many baneling	5e-4	ReLU	15	1	0.2	0.01	rnn	1	200	0.01
8m	5e-4	ReLU	15	1	0.2	0.01	rnn	1	200	0.01
MMM	5e-4	ReLU	15	1	0.2	0.01	rnn	1	200	0.01
1c3s5z	5e-4	ReLU	15	1	0.2	0.01	rnn	1	200	0.01
8m vs. 9m	5e-4	ReLU	15	1	0.05	0.01	rnn	1	200	0.01
bane vs. bane	5e-4	ReLU	15	1	0.2	0.01	rnn	1	200	0.01
25m	5e-4	ReLU	10	1	0.2	0.01	rnn	1	200	0.01
5m vs. 6m	5e-4	ReLU	10	1	0.05	0.01	rnn	1	200	0.01
3s5z	5e-4	ReLU	5	1	0.2	0.01	rnn	1	200	0.01
MMM2	5e-4	ReLU	5	2	0.2	1	rnn	1	200	0.01
10m vs. 11m	5e-4	ReLU	10	1	0.2	0.01	rnn	1	200	0.01
3s5z vs. 3s6z	5e-4	ReLU	5	1	0.2	0.01	rnn	1	200	1
27m vs. 30m	5e-4	ReLU	5	1	0.2	0.01	rnn	1	200	1
6h vs. 8z	5e-4	ReLU	5	1	0.2	0.01	mlp	1	200	1
corridor	5e-4	ReLU	5	1	0.2	0.01	mlp	1	200	1

Table 14: Adopted hyperparameters used for MAPPO and QMix in the SMAC domain.

Tasks	MAPPO						
	lr	epoch	mini-batch	activation	gain	entropy coef	network
2-player	actor:7e-4 critic:1e-3	15	1	ReLU	0.01	0.015	mlp

Table 15: Adopted hyperparameters used for MAPPO in the Hanabi domain.

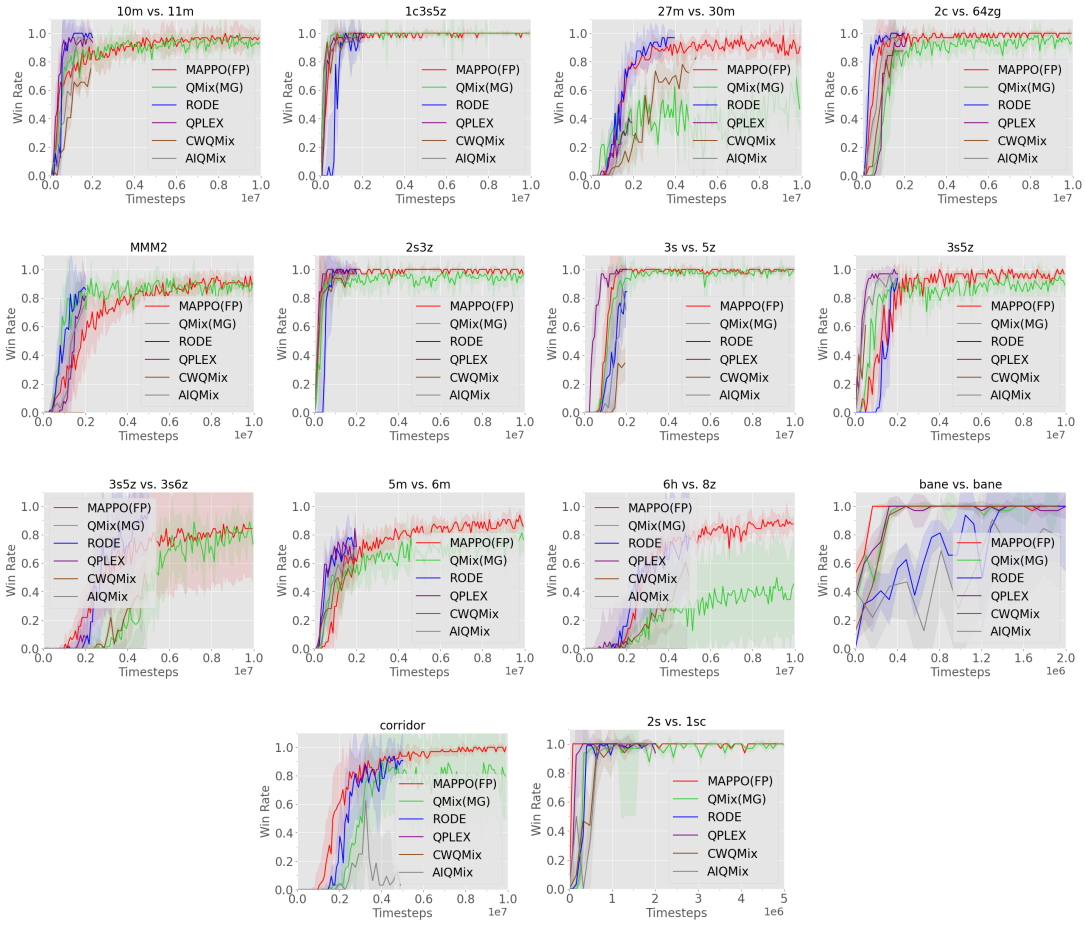


Figure 15: Median evaluation win rate of MAPPO(FP), QMix(MG), RODE, QPlEX, CWQMix and AIQMix in the SMAC domain.

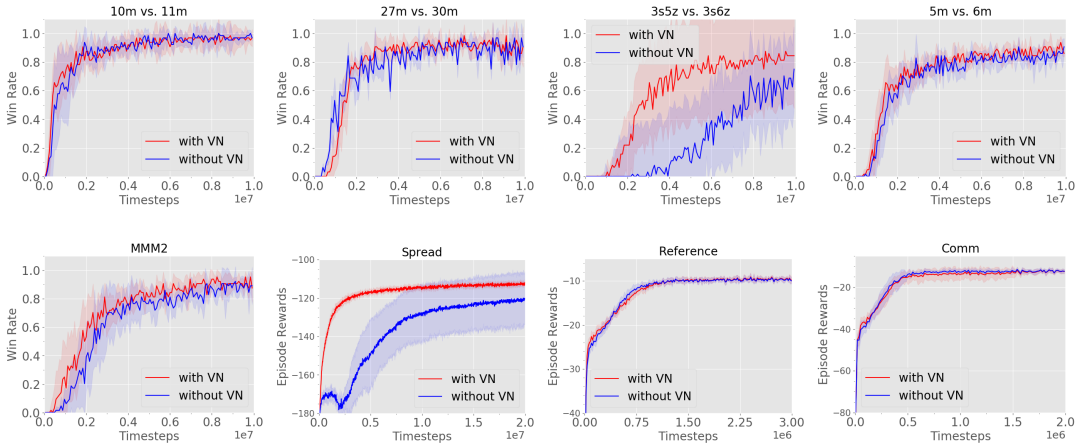


Figure 16: Ablation studies demonstrating the effect of Value Normalization(VN) on MAPPO's performance in the SMAC and MPE domains.

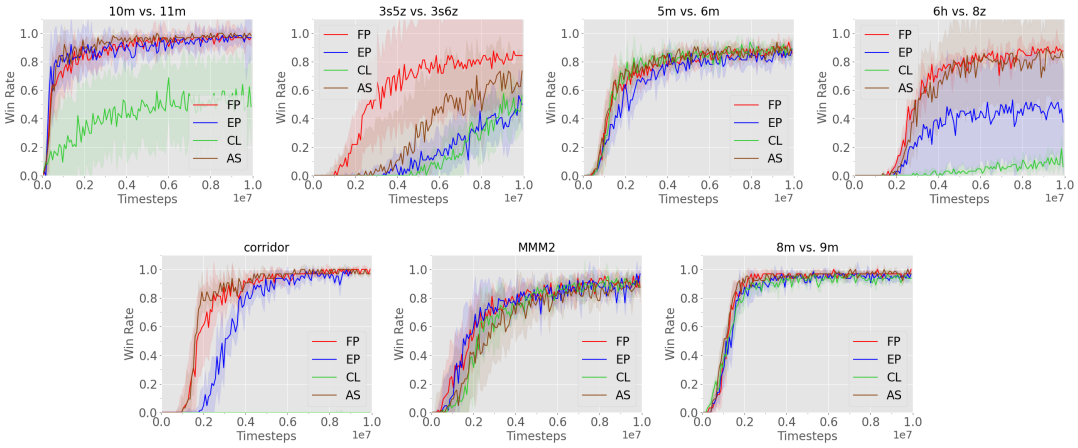


Figure 17: Ablation studies demonstrating the effect of different global state on MAPPO's performance in the SMAC domain.

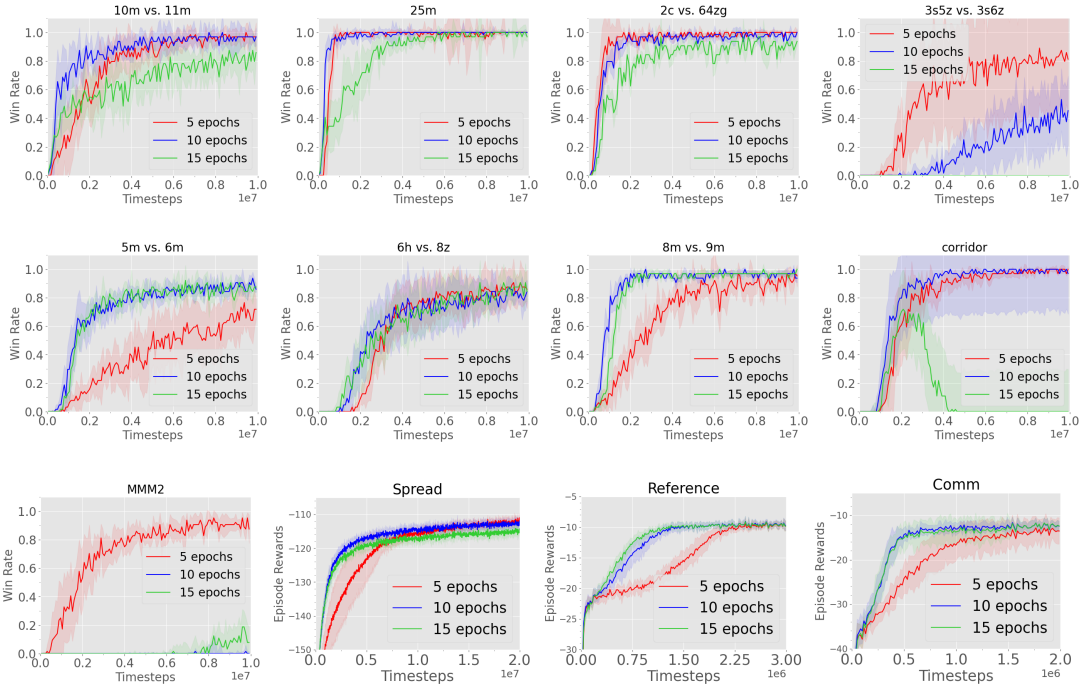


Figure 18: Ablation studies demonstrating the effect of training epochs on MAPPO's performance in the SMAC and MPE domains.

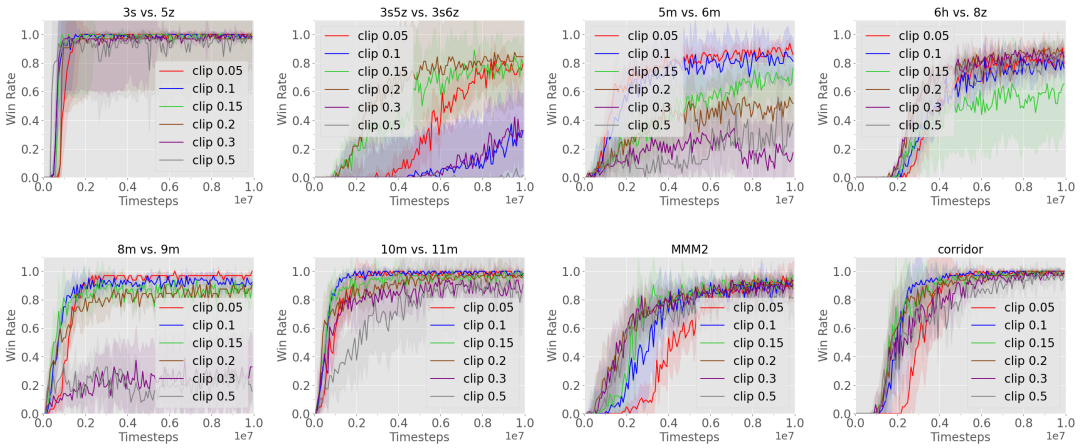


Figure 19: Ablation studies demonstrating the effect of clip term on MAPPO's performance in the SMAC domain.

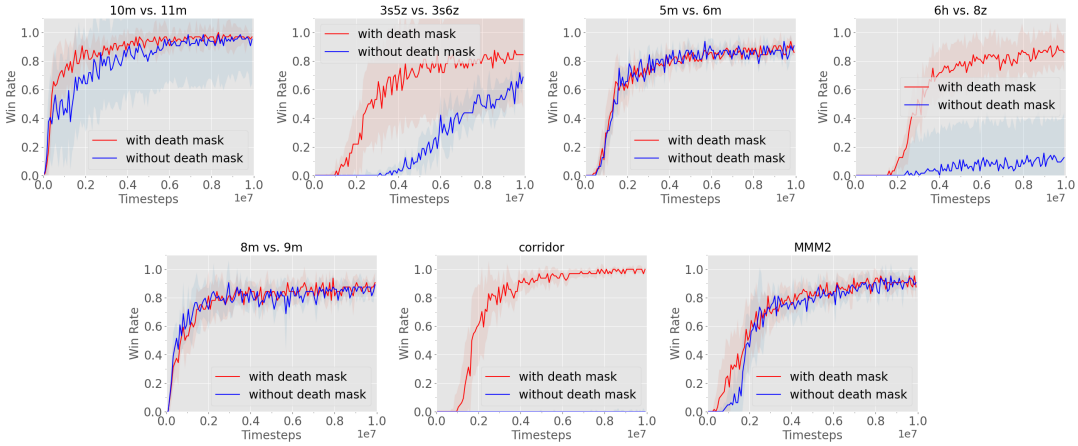


Figure 20: Ablation studies demonstrating the effect of death mask on MAPPO(FP)'s performance in the SMAC doamin.

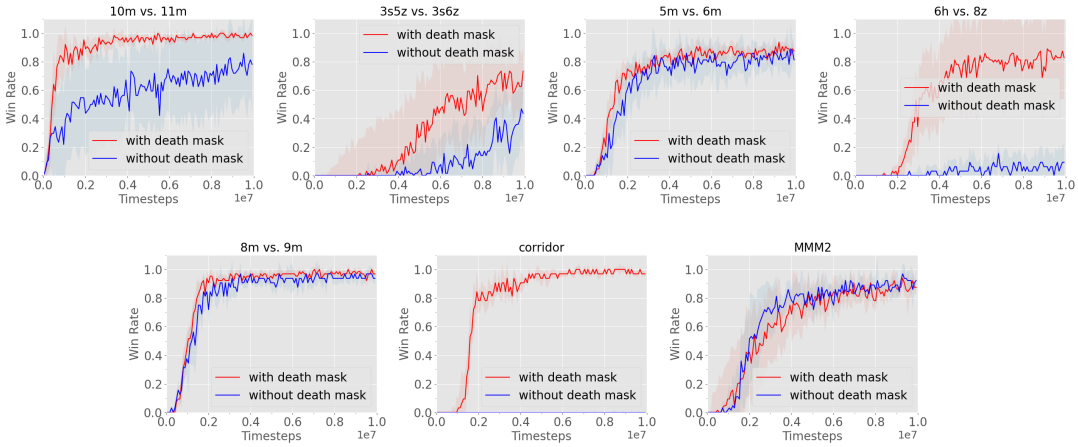


Figure 21: Ablation studies demonstrating the effect of death mask on MAPPO(AS)'s performance in the SMAC domain.

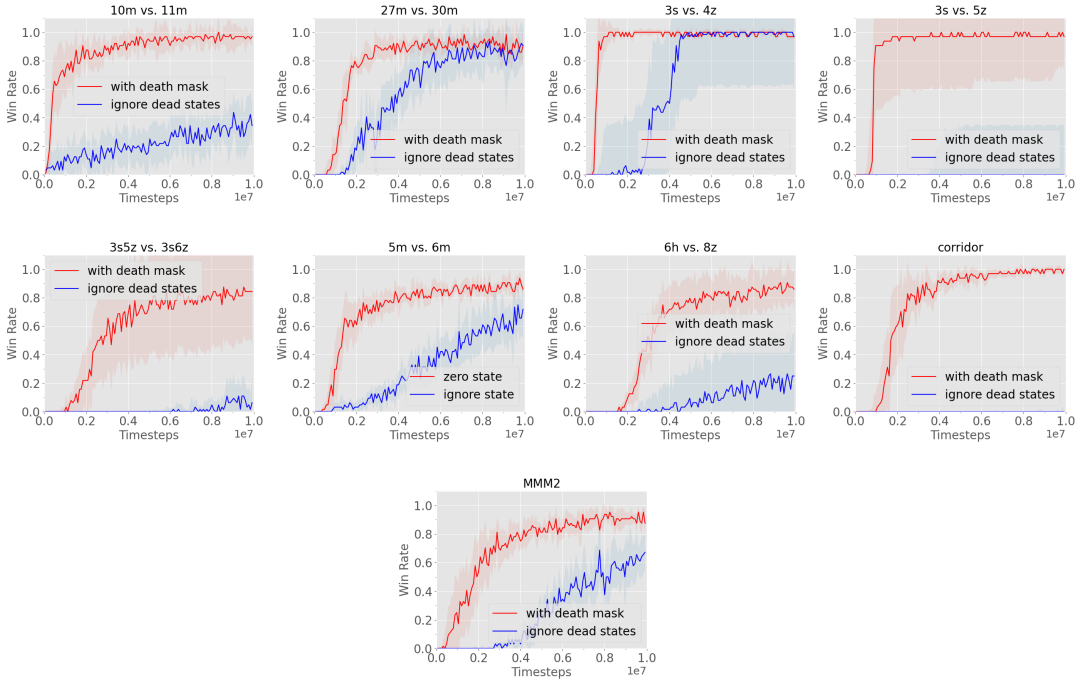


Figure 22: Ablation studies demonstrating the effect of death mask on MAPPO's performance in the SMAC domain.

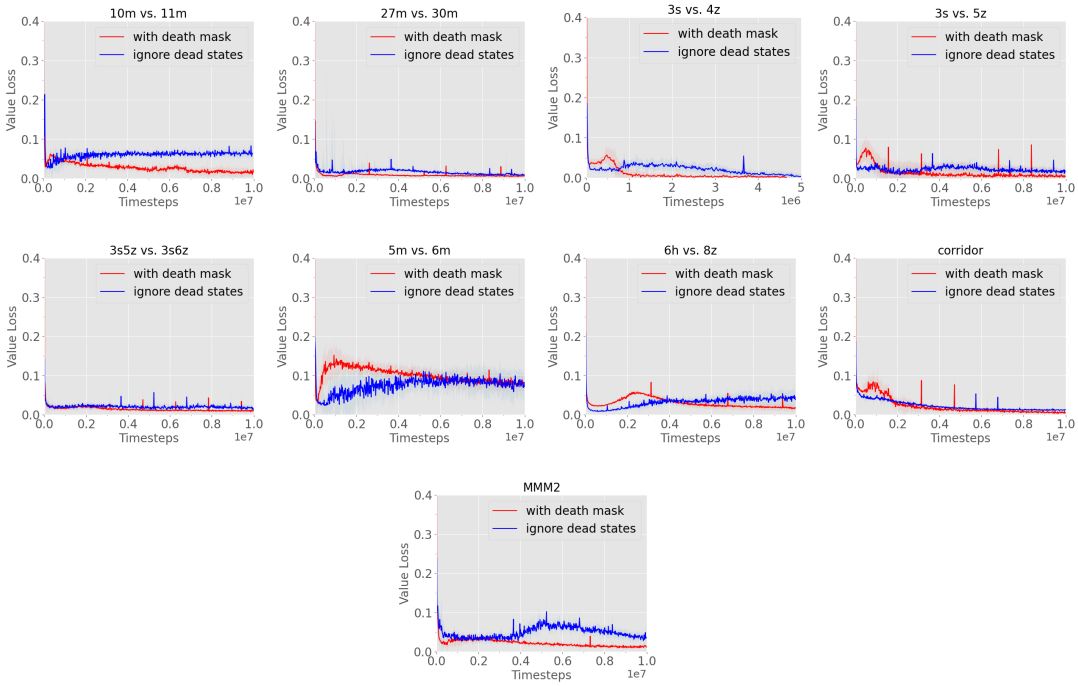


Figure 23: Effect of death mask on MAPPO's value loss in the SMAC domain.

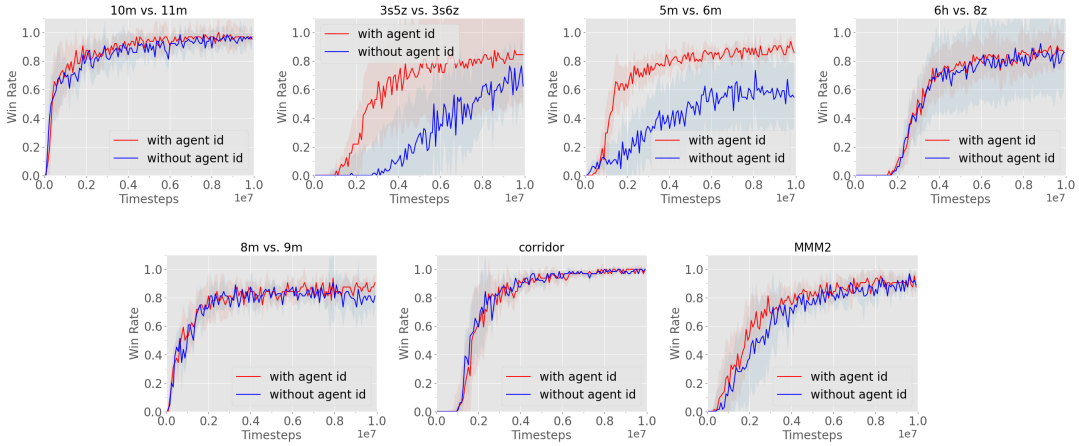


Figure 24: Ablation studies demonstrating the effect of agent id on MAPPO's performance in the SMAC domain.