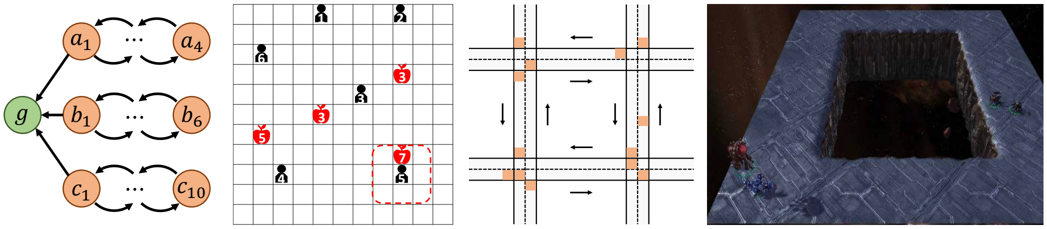
A Appendix

A.1 Details about Benchmarks

We introduce four types of testing environments as shown in Fig. 1 in our paper, including Hallway [6], Level-Based Foraging (LBF) [2], Traffic Junction (TJ) [1], and two maps named 1o2r_vs_4r and 1o10b_vs_1r requiring communication from StarCraft Multi-Agent Challenge (SMAC) [6]. In this part, we will describe the details of these environments used.



(a) Hallway (b) LBF (c) Traffic Junction (TJ) (d) SMAC

Figure 1: Multiple benchmarks used in our experiments.

Hallway We design two instances of the Hallway environment. In the first instance, we apply three hallways with lengths of $4, 6$, and 10, respectively. That means we let three agents a, b, c respectively initialized randomly at states a_1 to a_4, b_1 to b_6, and c_1 to c_{10}, and require them to arrive at state g simultaneously. In the second instance, we divide 5 agents into two groups. The first group has hallways with lengths of 3 and 5, and the second group has hallways with lengths of $4, 6$, and 10. A reward of 1 will be given if one group arrives at the goal g simultaneously. However, if both groups reach the goal simultaneously, a penalty of -0.5 will be given.

Level-Based Foraging (LBF) We use a variant version of the original environment used in [2], where we define the `state` to be a data structure that can represent the true global state instead of concatenating the observations of all agents directly. On this basis, we use two environment instances with different configurations, of which one is an 11×11 grid world with 6 agents, 4 foods, and the other is a 20×20 grid world with 10 agents, 6 foods. In both instances, the observation of agents is a 3×3 field of view around it.

Traffic Junction (TJ) We use the *medium* and *hard* versions of the Traffic Junction environments. The $medium$ version has an agent number limit of 10, and the road dimension is 14, while the $hard$ version has an agent number limit of 20, and the road dimension is 18. In both of these two instances, the sight of the agent is limited to 0, which means each agent can only observe a 1×1 field of view around it.

StarCraft Multi-Agent Challenge (SMAC) We use two maps named 1o2r_vs_4r and 1o10b_vs_1r in SMAC, which are introduced in NDQ [6]. In 1o2r_vs_4r, an Overseer finds 4 Reapers, and the ally units, 2 Roaches, need to reach enemies and kill them. Similarly, 1o10b_vs_1r is a map full of cliffs, where an Overseer detects a Roach, and the randomly spawned ally units, 10 Banelings, are required to reach and kill the enemy.

A.2 Details about the algorithms involved in paper

Several algorithms are involved in our work, including Full-Comm, NDQ [6], TarMAC [1], TMC [8] and QMIX [3]. All these methods follow the setting of Dec-POMDP in our experiments, which means that each agent can only have access to its individual partial observation at each timestep. Some algorithms among them (Full-Comm, NDQ, TarMAC, TMC) will do communication to base each agent's decision-making on richer information, while QMIX has no communication and always let agents make decisions based on their local observations (or observation histories). In Table 1, we offer a comparison between these baselines and our method from different dimensions.

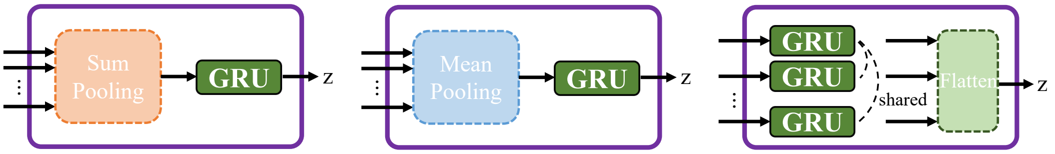
Table 1: Comparison of various algorithms used in this paper

Name	Type of communication	Where to process the information (the Sender/Receiver)	Matched scenarios
MASIA(ours)	Full	Receiver	No Restrictions
Full-Comm	Full	No	Without redundant information
NDQ	Full	Sender	Value function is nearly decomposable
TarMAC+QMIX	Full	Receiver	Message with relative importance
TMC	Time-Partial	Sender & Receiver	Message with transmission loss
QMIX	No	No	Full observation or easy coordination

A.3 Implementation Details

A.3.1 Network Architecture and Hyper-parameters

Integration Network The Information Aggregation Encoder (IAE) in our approach consists of a Self-Attention Network and an Integration Network. Here we describe the details of the Integration Network. We introduce 3 different kinds of integration networks as shown in Fig. 2.



(a) (b) (c)

Figure 2: Three versions of implementation for integration network. The dotted lines in (c) indicate that these GRU networks share the same parameters.

Among these three versions of integration networks, (a) and (b) apply *mean* and *sum* pooling operations, respectively, on the output vectors of the self-attention network, while in the third version (c), we let the self-attention network's output vectors pass through a shared GRU network separately and finally flatten them as the obtained aggregation representation. In (a) and (b), no matter how we permute the agents, we will always obtain the same aggregation representation. In (c), the permutation of the agents will only affect the order of some dimensions of the aggregation representation instead of obtaining a totally different representation in some vanilla designs, such as networks with Multi-Layer Perceptions. In practice, we find that design (c) achieves the best performance among these three integration networks, and all the experiment results shown in our paper are based on this implementation.

Hyper-parameters Our implementation of MASIA is based on the EPyMARL[1] [4] with StarCraft 2.4.6.2.69232 and uses its default hyper-parameter settings. For example, we apply the default ϵ-greedy action selection algorithm to each method, which means ϵ decays from 1 to 0.05 in 50K timesteps. The selection of the additional parameters introduced in our approach is listed in Table 2. We use this set of parameters in all experiments shown in this paper except for the ablations.

A.3.2 Experimental Details

Our experiments were performed on a desktop machine with 4 NVIDIA GTX 3090 GPUs. For all the performance curves in our paper, we pause training every M timesteps and evaluate for N episodes with decentralized greedy action selection. The (M, N) in Hallway [6], Level-Based Foraging [2], Traffic Junction [1] and SMAC [6], are $(10\mathrm{K}, 100)$, $(50\mathrm{K}, 100)$, $(10\mathrm{K}, 40)$ and $(50\mathrm{K}, 100)$, respec-

---

[1] `https://github.com/uoe-agents/epymarl`

Table 2: Hyper-parameters in experiments

name	value
hidden dimension for query and key in self-attention module	16
output dimension of self-attention module	32
λ_2 (coefficient of latent model learning loss)	1
λ_1 (coefficient of encoder-decoder learning loss)	1
action embedding dimension in latent model	8
dimension for each agent in the aggregation representation	8
observation embedding dimension before concatenated with z	32
whether to predict the residuals of the next state	True
prediction length K in latent model	2
hidden dimension for hidden states in latent model	64

tively. We evaluate the test win rate, the percentage of episodes in which the agents win the game within the time limit in N testing episodes for all tasks.

A.4 Sensibility of Prediction Length K

To study the function of the second self-supervised objective we designed, we apply different latent model prediction lengths K in the task of TJ (medium). The learning results of the beginning 0.3M samples are shown in Fig. 3. From the learning curves of different K, we can find that algorithms applying a latent model learning objective ($K > 0$) can learn faster than that without this objective ($K = 0$). Besides, although our experiments in the main text uniformly set the prediction length K to 2, we find that the algorithm learns fastest when K is set to 4, which indicates that fine-tuning the hyper-parameters of the self-supervised learning can further advance the performance of MASIA.

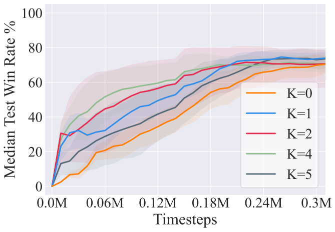


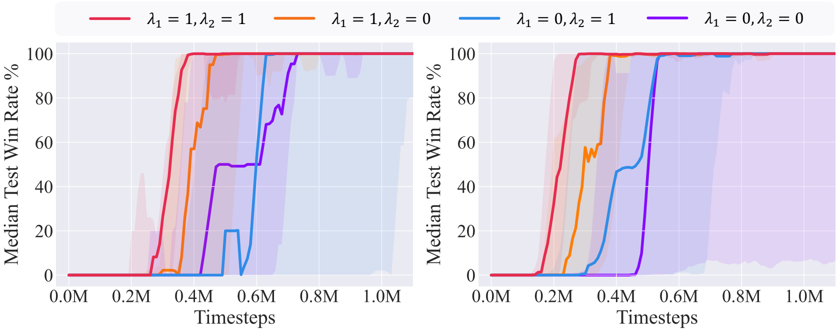
Figure 3: Ablation study for the prediction length K.

A.5 Ablation for Representation Objectives

In our work, we propose two representation objectives to make the aggregated information representation *compact* and *sufficient*. To further justify the effectiveness of these two objectives, we conduct ablations for the hyper-parameter λ_1 and λ_2. Specifically, we compare with three ablations: (1)$\lambda_1 = 1, \lambda_2 = 0$; (2)$\lambda_1 = 0, \lambda_2 = 1$; (3)$\lambda_1 = 0, \lambda_2 = 0$, which respectively correspond to (1) only use encoder-decoder learning loss; (2) only use latent model learning loss; (3) neither loss is used. While our method (MASIA) corresponds to $\lambda_0 = 1, \lambda_1 = 1$. We conduct the ablation experiments in the tasks of Hallway and medium level TJ, and run each ablation for 5 random seeds. The experimental results for Hallway are illustrated in Fig. 4 and those for TJ (medium) are listed in Table 3. From both of these experimental results, we can see that MASIA with two representation objectives outperforms all ablations. In the task of Hallway, MASIA can learn to solve the task faster in both settings of 4x6x10 and 3x5-4x6x10. Especially, some random seeds of the third ablation fail to solve the task 3x5-4x6x10 within 2M samples, which shows the indispensable roles of these two

Table 3: Ablation experiments in the task of TJ (medium).

	1M	2M	4M
$\lambda_1 = 0, \lambda_2 = 1$	0.8125 ± 0.0473	0.8667 ± 0.0514	0.8950 ± 0.0292
$\lambda_1 = 1, \lambda_2 = 0$	0.7750 ± 0.0652	0.8550 ± 0.0534	0.9200 ± 0.0430
$\lambda_1 = 0, \lambda_2 = 0$	0.8000 ± 0.0204	0.8583 ± 0.0624	0.9167 ± 0.0312
$\lambda_1 = 1, \lambda_2 = 1$	$\mathbf{0.8938} \pm 0.0480$	$\mathbf{0.9000} \pm 0.0637$	$\mathbf{0.9225} \pm 0.0375$



(a) Hallway: 4x6x10

(b) Hallway: 3x5-4x6x10

Figure 4: Ablation experiments in the tasks of Hallway.

representation objectives. Similarly, in the task of TJ (medium), we compare the ablations' average test won rate performance at 1M, 2M, and 4M timesteps, respectively. Although three ablations show similar convergence performance to MASIA, MASIA achieves the best performance when at 1M timestep. This demonstrates that the proposed two representation objectives offer good guides and accelerate the task learning.

Algorithm 1 Overall Training Framework

```
Initialize replay buffer $\mathcal{D}$.
Initialize information aggregation encoder with random parameters $\theta$ and state prediction decoder with random parameters $\eta$.
Initialize $Q$ network with random parameters $\phi$ and latent model with random parameters $\psi$.
Initialize parameters of target encoder $\theta^- = \theta$, and target $Q$ network $\phi^- = \phi$.
for episode $= 1$ to $M$ do
    Roll out one trajectory $\boldsymbol{\tau}$ with $\epsilon$-greedy policy in the environment.
    Store the trajectory $\boldsymbol{\tau}$ in $\mathcal{D}$.
    if $|\mathcal{D}|$ is larger than batch size $m$ then
        Sample a minibatch $\mathcal{B}$ of $m$ trajectories from $\mathcal{D}$.
        Compute the encoder-decoder loss:
            $$\mathcal{L}_{ae}(\theta, \eta) = \sum_{\text{traj} \in \mathcal{B}} \sum_{t=1}^{T} \| g_\eta(z^t) - s^t \|_2^2, \ z^t = f_\theta(\boldsymbol{o}^t).$$
        Compute the latent model loss:
            $$\mathcal{L}_m(\theta, \psi) = \sum_{\text{traj} \in \mathcal{B}} \sum_{t=1}^{T-K} \sum_{k=1}^{K} \| \hat{z}^{t+k} - z^{t+k} \|_2^2,$$
            $$\hat{z}^{k+1} = h_\psi(z^t, \boldsymbol{a}^t), \ \hat{z}^{t+k} = h_\psi(\hat{z}^{t+k-1}, \boldsymbol{a}^{t+k-1}), k = 2, \ldots, K,$$
            $$z^{t+k} = f_{\theta^-}(\boldsymbol{o}^{t+k}), k = 0, \ldots, K,$$
        Compute the reinforcement learning loss:
            $$\mathcal{L}_{rl}(\theta, \phi) = \sum_{\text{traj} \in \mathcal{B}} \sum_{t=1}^{T-1} \left( r + \gamma \max_{\boldsymbol{a}'} Q_{\text{tot}}(\boldsymbol{\tau}^{t+1}, \boldsymbol{a}'; \theta^-, \phi^-) - Q_{\text{tot}}(\boldsymbol{\tau}^t, \boldsymbol{a}^t; \theta, \phi) \right)^2.$$
        Update $\theta, \eta, \phi, \psi$ by minimizing $\mathcal{L}_{rl}(\theta, \phi) + \lambda_1 \mathcal{L}_{ae}(\theta, \eta) + \lambda_2 \mathcal{L}_m(\theta, \psi)$.
    end if
    Update target network parameters $\theta^-, \phi^-$ with the EMA method.
end for
```

A.6 The Overall Flow of MASIA for Training and Testing

Firstly, to offer a direct impression of our work, we first talk about how agents behave differently during decentralized execution vs. centralized training in our method (MASIA). The main differences are that we would use state information to help compute the auto-encoder loss and estimate the Q_{tot} (if there is a mixing network in the Q-network, e.g. MASIA+QMIX) during training phase. Besides, multi-step prediction loss is also only computed and optimized in the training phase. During decentralized execution, the state decoder, the latent model and the possible mixing network are all thrown away. We remain the Information Aggregation Encoder, the focusing network and individual Q-networks to ensure the decentralized execution process. The representation loss terms we designed are aimed at training the encoder network well.

To illustrate the process of training, the overall training flow of MASIA is shown in Alg. 1. Lines 5$\sim$16 express the whole training process, where we apply an off-policy learning algorithm and iteratively update the parameters of the model. Specifically, we compute encoder-decoder loss, latent model loss and temporal difference loss in Lines 10, 11 and 12, respectively, and do parameter updating together in Line 13.

Besides, the execution flow of MASIA is shown in Alg. 2. In the execution phase, the agents first broadcast their observations to each other, and then each agent calculates its own action a_i^t by applying the focusing network and individual Q network learned during training, which is described in Lines 4 and 5.

Algorithm 2 Overall Execution Flow

```
Require: information aggregation encoder with parameters $\theta$, individual $Q$ network with parameters $\phi_i$ for each $Q_i$, focusing networks with parameter $\omega_i$ for each agent and agent number $n$.
1: for step $= 0$ to episode_limit do
2:     Each agent broadcasts its observational information $o_i^t$ at timestep $t$, and then each agent feed collected observations $\boldsymbol{o}^t = \{o_i^t\}_n$ into the information aggregation network, obtaining $z^t = f_\theta(\boldsymbol{o}^t)$.
3:     for $i = 1$ to $n$ do
4:         Agent $i$ calculates $w_i^t = F_{w_i}(o_i^t)$ by using the focusing network, and obtains extracted information $\bar{z}_i^t = w_i^t \cdot z^t$.
5:         The $o_i^t$ and $\bar{z}_i^t$ are fed into the individual $Q$ network to calculate $Q_i(\tau_i^t, \cdot)$, and agent $i$ gets action $a_i^t = \arg\max_a Q_i(\tau_i^t, a)$.
6:     end for
7:     The agent system interacts with the environment by executing actions $\{a_i^t\}_n$.
8: end for
```

A.7 Limitations and Possible Negative Societal Impacts

In our work, the agents need to full-communicate their individual observational information first before doing further information aggregation. This may be vulnerable when facing communication attacks among agents. There exist some recent research works [5, 7] concerning adversarial attacks, but the discussions about the problem in multi-agent communication reinforcement learning are still limited. How to obtain a robust and efficient communication protocol is of great value. In short, broader negative societal impacts have not been found at this stage, and how to handle the possible problems we believe will be an interesting direction for research.

References

[1] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. TarMAC: Targeted multi-agent communication. In *International Conference on Machine Learning*, pages 1538–1546, 2019.

[2] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[3] 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 *International Conference on Machine Learning*, pages 4295–4304, 2018.

[4] Mikayel Samvelyan, Tabish Rashid, Christian Schröder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob N Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *International Conference on Autonomous Agents and MultiAgent Systems*, pages 2186–2188, 2019.

[5] Yanchao Sun, Ruijie Zheng, Parisa Hassanzadeh, Yongyuan Liang, Soheil Feizi, Sumitra Ganesh, and Furong Huang. Certifiably robust policy learning against adversarial communication in multi-agent systems. *arXiv preprint arXiv:2206.10158*, 2022.

[6] Tonghan Wang, Jianhao Wang, Chongyi Zheng, and Chongjie Zhang. Learning nearly decomposable value functions via communication minimization. In *International Conference on Learning Representations*, 2020.

[7] Wanqi Xue, Wei Qiu, Bo An, Zinovi Rabinovich, Svetlana Obraztsova, and Chai Kiat Yeo. Misspoke or mis-lead: Achieving robustness in multi-agent communicative reinforcement learning. In *International Conference on Autonomous Agents and MultiAgent Systems*, pages 1418–1426, 2022.

[8] Sai Qian Zhang, Qi Zhang, and Jieyu Lin. Succinct and robust multi-agent communication with temporal message control. In *Advances in Neural Information Processing Systems*, pages 17271–17282, 2020.