

## A RELATED WORKS

This section includes the related works about network designs in cooperative MARL, state space models for sequence modeling and self-predictive RL methods.

### A.1 NETWORK DESIGN IN MARL

Early MARL algorithms adopted MLP-based policy and value networks which are commonly used in single-agent RL algorithms (Lowe et al., 2017; Sunehag et al., 2017). However, when dealing with partially observable tasks, the necessity of utilizing historical information becomes non-negligible (Gupta et al., 2017). The most popular recurrent network structure is GRU. COMA utilizes GRU in the actor network and MLP consisting multiple layers with ReLU as the critic network (Foerster et al., 2018). All agents in QMIX comprise a GRU as the core part, as well as its various variants (Rashid et al., 2020; Yang et al., 2020; Wang et al., 2020; Jeon et al., 2022; Wang et al., 2023). SOTA policy-based MARL algorithms also widely adopt GRUs as policy and critic networks (Yu et al., 2022; Hu et al., 2021).

### A.2 STATE SPACE MODELS FOR SEQUENCE MODELING

Structured state space models (S4 models) are already proven to be sufficient in sequence signal compression, audio generation, image and audio modeling (Gu et al., 2020; 2021; 2022b; Orvieto et al., 2023; Smith et al., 2023; Nguyen et al., 2022), etc. Various works have improved S4 models through modifying the way of parameterization and initialization (Gu et al., 2022a; Gupta et al., 2022). S4 models have also shown promising performance in partially-observable RL (Lu et al., 2023), in-context RL (Chen et al., 2023), imitation learning (Jia et al., 2024) and world modeling (Deng et al., 2023). Mamba is a new type of S4 model with selection and a scan (Gu & Dao, 2023) which demonstrates superior performance compared to Transformers with similar scales. Recently, Mamba has garnered widespread attention due to its strong long-sequence reasoning and modeling capabilities, coupled with its linearly scalable computational resource consumption.

### A.3 SELF-PREDICTIVE RL

Self-predictive representation learning methods have shown noticable data efficiency in sequence modeling tasks like vision and language processing, especially in low data regimes (Xie et al., 2020; Henaff, 2020; Chen et al., 2020). Inspired from VAE, self-predictive representation learning is recently widely used in world modeling (Łukasz Kaiser et al., 2020; Hafner et al., 2021; Zhang et al., 2023; Micheli et al., 2023; Robine et al., 2023; Hansen et al., 2024a;b) to extract relevant features for transition prediction. Through reconstructing the raw input and predicting the future latents or inputs, world models can learn complicated environment dynamics for training competitive agents with small amount of data. Self-predictive representation learning can also improve data efficiency in deep Q-learning (Schwarzer et al., 2020; 2021; 2023). Given the success self-predictive RL methods have achieved, we believe it is promising implementing self-predictive representation learning objectives in policy learning.

## B MAIN COMPARISONS AGAINST BASELINES

We present full learning curves about the comparison among self-predictive Mamba, vanilla Mamba and GRU in Section 5.2 here. We also list the detailed hyperparameters of the comparison.

## B.1 FULL RESULTS

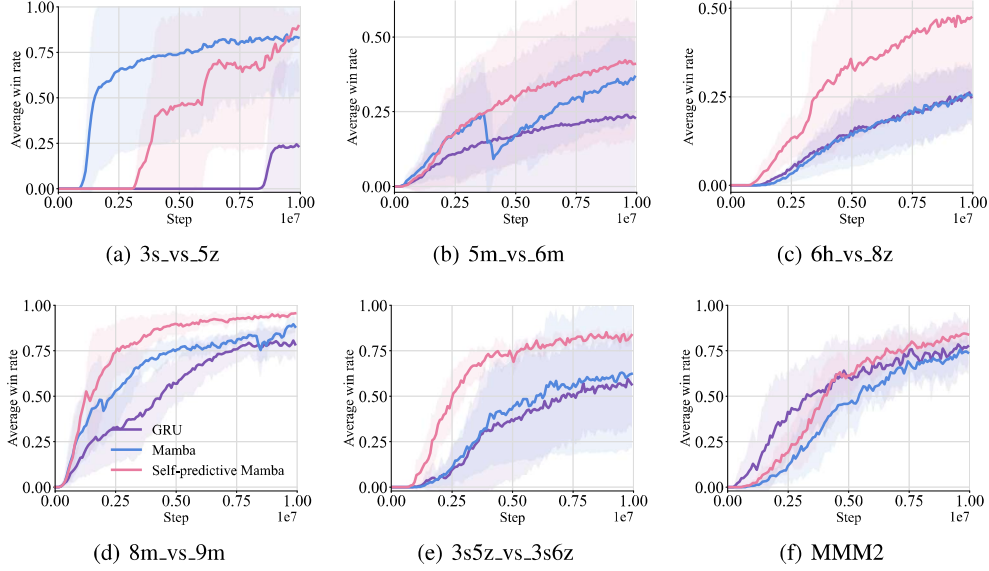


Figure 1: Comparison between the self-predictive Mamba, RNN, and the vanilla Mamba policies.

## B.2 HYPERPARAMETERS OF SELF-PREDICTIVE MAMBA, MAMBA AND GRU

We use the implementation <https://github.com/marlbenchmark/on-policy> for GRU policy in MAPPO (Yu et al., 2022). We use the Mamba implementation <https://github.com/state-spaces/mamba> for building Mamba policy. We share the same hyperparameter setting among all the 6 tasks. Additionally, as self-predictive Mamba is an extension of Mamba, we maintain identical values for the shared parameters between Mamba and self-predictive Mamba.

Table 1: Hyperparameters of self-predictive Mamba and GRU policy.

Parameter name	self-predictive Mamba	GRU
Input sequence length $k$	1	-
Output dimension $d$	64	-
Expansion coefficient $\epsilon$	2	-
Latent state dimension	128	64
Weight coefficient $\alpha$	0.1	-
Clip parameter $\eta$	0.2	0.2
Early stop coefficient $\mu$	0.05	-
GAE Discount coefficient $\gamma$	0.99	0.99
GAE $\lambda$	0.95	0.95
Optimizer	Adam	Adam
Actor learning rate	5e-4	5e-4
Critic learning rate	5e-4	5e-4
Adam $\epsilon$	1e-5	1e-5
Gradient clipping	10	10
PPO epoch number	5	5
Batch size	3200	3200
Seed	1, 2, 3, 4	1, 2, 3, 4

Table 2: Running time of 6 tasks.

Task name	Running time (h)
3s_vs_5z	25.3
5m_vs_6m	22.7
8m_vs_9m	22.6
6h_vs_8z	31.46
3s5z_vs_3s6z	24.6
MMM2	32

### B.3 HYPERPARAMETERS OF RODE

We use the implementation <https://github.com/TonghanWang/RODE> for executing RODE (Wang et al., 2021). We share the same hyperparameters among all the 6 tasks.

Table 3: Hyperparameters of RODE.

Parameter name	RODE
Action latent dim	20
Batch size	32
Buffer size	5000
Learning rate	5e-4
$\epsilon$ -greedy	1.0→0.05
$\epsilon$ anneal time	70000
Reward discount coefficient $\gamma$	0.99
Hypernet embedding dimension	64
Hypernet layer number	2
Mixing embedding dimension	32
Role cluster number	3
Role action spaces update start	5000
Role $\epsilon$ finish	0.05
Role updating interval	5
Target network update interval	200
State latent dimension	32
Optimizer	RMSProp
RMSProp $\alpha$	0.99
RMSProp $\epsilon$	1e-5
Gradient clipping	10
Latent state dimension	64
Seed	1, 2, 3, 4

### B.4 HYPERPARAMETERS OF QMIX

We use the implementation <https://github.com/marlbenchmark/off-policy> for executing QMIX (Rashid et al., 2020). We share the same hyperparameters among all the 6 tasks.

Table 4: Hyperparameters of QMIX.

Parameter name	QMIX
Batch size	32
Buffer size	5000
Learning rate	5e-4
$\epsilon$ -greedy	1.0 $\rightarrow$ 0.05
$\epsilon$ anneal time	50000
Reward discount coefficient $\gamma$	0.99
Hypernet embedding dimension	64
Hypernet layer number	2
Mixing embedding dimension	32
Target network update interval	200
Target network update interval (episode)	200
Number of episodes to add to buffer with purely random actions	5
Training interval	100
Training interval (episode)	1
Optimizer	RMSProp
RMSProp $\alpha$	0.99
RMSProp $\epsilon$	1e-5
Gradient clipping	10
Latent state dimension	64
Seed	1, 2, 3, 4

## C ENCODER DESIGNING

We present the detailed self-predicting loss curves of the comparison among MLP-VAE, categorical-VAE and SimNorm-VAE in Section 5.3.1 here.

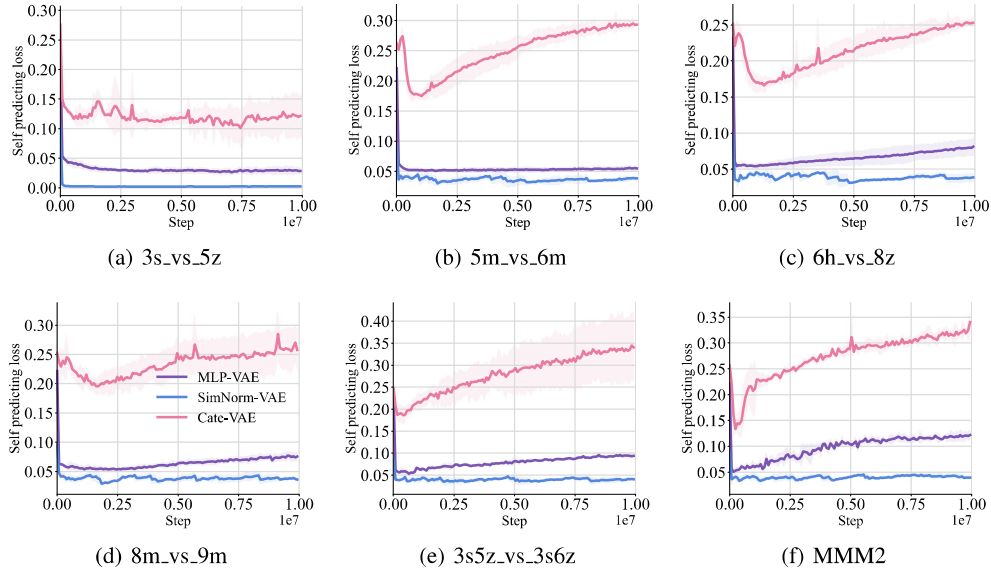


Figure 2: The self-predicting losses for MLP-VAE, categorical-VAE and SimNorm-VAE.