Multi-Agent Decision S4: Leveraging State Space Models for Offline Multi-Agent Reinforcement Learning

[1]**Ashmita Bhattacharya**, [2]**Malyaban Bal**

The Pennsylvania State University
University Park, PA 16801
[1]azb6481@psu.edu, [2]mjb7906@psu.edu

Abstract

Goal-conditioned sequence-based supervised learning with transformers has shown promise in offline reinforcement learning (RL) for single-agent settings. However, extending these methods to offline multi-agent RL (MARL) remains challenging. Existing transformer-based MARL approaches either train agents independently, neglecting multi-agent system dynamics, or rely on centralized transformer models, which face scalability issues. Moreover, transformers inherently struggle with long-term dependencies and computational efficiency. Building on the recent success of Structured State Space Sequence (S4) models, known for their parameter efficiency, faster inference, and superior handling of long context lengths, we propose a novel application of S4-based models to offline MARL tasks. Our method utilizes S4's efficient convolutional view for offline training and its recurrent dynamics for fast on-policy fine-tuning. To foster scalable cooperation between agents, we sequentially expand the decision-making process, allowing agents to act one after another at each time step. This design promotes bi-directional cooperation, enabling agents to share information via their S4 latent states or memory with minimal communication. Gradients also flow backward through this shared information, linking the current agent's learning to its predecessor. Experiments on challenging MARL benchmarks, including Multi-Robot Warehouse (RWARE) and StarCraft Multi-Agent Challenge (SMAC), demonstrate that our approach significantly outperforms state-of-the-art offline RL and transformer-based MARL baselines across most tasks.

Introduction

Multi-agent reinforcement learning (MARL) excels at learning complex, coordinated policies for shared objectives (Cao et al. 2012; Berner et al. 2019; Ye, Zhang, and Yang 2015) but often requires a large volume of costly or risky interactions with the environment. To improve sample efficiency, offline reinforcement learning (RL) algorithms (Lee et al. 2021; Fujimoto, Meger, and Precup 2019; Kumar et al. 2019, 2020; Kostrikov et al. 2021; Xu et al. 2022; Li et al. 2022; Xu et al. 2023) enable learning from pre-collected datasets, reducing the need for extensive online interactions.

Offline RL faces distribution shifts, causing extrapolation errors when policies encounter out-of-distribution (OOD) samples, as the learned policy deviates from the behavior policy used to collect the data. Regularization techniques (Kumar et al. 2019) help mitigate this by keeping the learned policy close to the behavior policy (Kumar et al. 2020; Xu et al. 2023, 2022). While efforts have been made to extend these methods to multi-agent settings (Shao et al. 2023; Wang et al. 2024; Pan et al. 2022; Yang et al. 2021), the joint state-action space grows exponentially with the number of agents, making global application of these regularizations challenging, especially with limited offline data.

On the other hand, sequence-based supervised learning, pioneered by the Decision Transformer (DT) (Chen et al. 2021), has also been applied in offline MARL by autoregressively predicting the next action based on the state, previous actions, and return-to-go. While extensions to offline MARL (Meng et al. 2021; Tseng et al. 2022) have been made, they still have limitations. MADT (Meng et al. 2021) adapts DT independently for each agent, ignoring cooperation, while (Tseng et al. 2022) uses a centralized teacher policy, which faces scalability issues. Moreover, both approaches suffer from transformer-specific drawbacks, such as large model sizes, inefficient inference, and difficulty capturing long-range dependencies due to fixed window constraints.

Structured State Space Sequence (S4) models (Gu, Goel, and Ré 2021) have recently outperformed transformer-based models in single-agent offline RL tasks (Bar-David et al. 2023). Building on this success, we propose a sequence-based offline MARL algorithm using S4 variants, which offer superior parameter efficiency and constant-time inference while capturing long contexts. Unlike MADT, which trains agents independently, our method explicitly models cooperation through a Sequentially Expanded MDP (SE-MDP) paradigm. In this framework, recently used in online MARL settings (Li et al. 2023), each decision step is divided into mini-steps, with agents acting sequentially based on their predecessors' actions. Unlike (Li et al. 2023), we enable minimal communication, requiring each agent to access only its immediate predecessor's information, shared through the latent state representation of the S4 model. Utilizing this hidden state of the S4 module of the current agent, information on all its prior agents is efficiently passed down to the next agent, and gradients flow backward from the current agent through this shared memory to the pre-

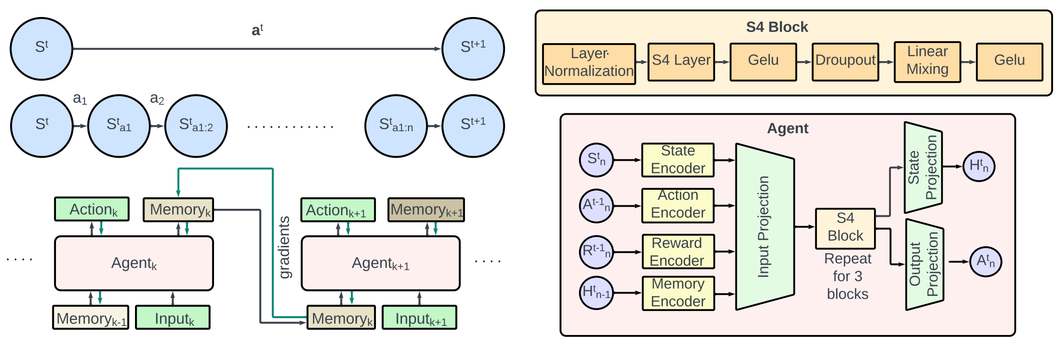

Figure 1: (Left) Multi-Agent MDP (MMDP) is restructured into a Sequentially-Expanded MDP (SE-MDP) where the multi-agent state transition at each timestep is decomposed into n intermediate states. Agents act sequentially, processing inputs with information from their predecessor, taking actions, and passing updated information forward. During training, the gradient flows backward, enabling earlier agents to receive updates based on the information passed by later agents. (Right) MADS4 Agent Actor Network. In addition to the state, action, and reward encoder, the memory/ S4 state representation from the previous agent (H_{n-1}) is encoded through a memory encoder. The updated S4 states and outputs from the S4 blocks pass through separate projection layers.

vious agents during training. This design enables scalable training with constant memory communication overhead, unlike traditional communication-based MARL algorithms, where memory overhead typically increases quadratically with the number of agents. This offline algorithm is well-suited for sample-efficient and scalable learning, making it highly applicable in real-world scenarios, including infrastructure management (Leroy et al. 2023), traffic flow optimization (Vinitsky et al. 2020), and more.

The S4-based agents are trained directly on sequences or trajectories from the offline dataset in an efficient convolutional manner. These pre-trained models can then be fine-tuned online using individual tuples, leveraging S4's recurrent dynamics. We evaluate the performance of our developed algorithm, called Multi-Agent Decision S4 (MADS4), on the challenging offline MARL benchmarks of Multi-Robot Warehouse (RWARE) (Papoudakis et al. 2020) and StarCraft2 Multi-Agent Challenge (SMAC) (Samvelyan et al. 2019), where MADS4 achieves superior performance across many tasks over state-of-the-art offline RL-based and transformer-based baselines.

Methodology

Sequentially Expanded MDP In this work, the Multi-agent Markov Decision Process (MMDP) is formulated as a Sequentially Expanded Markov Decision Process (SE-MDP), where each timestep is divided into n mini timesteps, with one agent acting during each mini-timestep. Thus, a single-step transition in the original MMDP $(s^t, \boldsymbol{a}^t, s^{t+1})$ resulting in a shared reward $r(s^t, \boldsymbol{a}^t)$ is decomposed into a sequence of n intermediate transitions, which collectively lead to the same shared reward $r(s^t, \boldsymbol{a}^t)$, as illustrated in Figure 1.

$$(s^t, \boldsymbol{a}^t, s^{t+1}) = \{(s^t, a_1^t, s_{a_1}^t), (s_{a_1}^t, a_2^t, s_{a_{1:2}}^t), ..., (s_{a_{1:n-1}}^t, a_n^t, s_{a_{1:n}}^t = s^{t+1})\} \tag{1}$$

Within this framework, at each timestep, an agent's action depends on information passed by its immediate predecessor in the sequence. This establishes a bidirectional dependency among agents, as illustrated in Figure 1: in the forward direction, an agent's actions are influenced by its predecessors, while in the backward direction, gradients propagate from the current agent to previous agents. While our algorithm is primarily developed within an SE-MDP framework, it is not confined to this setting, as we elaborate further in the discussion.

S4-based Agent At each time step, the model takes $u(t)$ as input and updates the latent state/memory $x(t)$ and, in turn, returns an output $y(t)$ by following the first-order differential equation:

$$\begin{aligned} \dot{x}(t) &= \boldsymbol{A}x(t) + \boldsymbol{B}u(t) \\ y(t) &= \boldsymbol{C}x(t) + \boldsymbol{D}u(t) \end{aligned} \tag{2}$$

The above continuous SSM can be discretized with a fixed step size Δ following any discretization scheme, such as the bilinear method (Tustin 1947), to obtain the following discretized linear recurrence relations:

$$\begin{aligned} x_k &= \bar{\boldsymbol{A}}x_{k-1} + \bar{\boldsymbol{B}}u_k(t) \\ y_k &= \bar{\boldsymbol{C}}x_k + \bar{\boldsymbol{D}}u_k \end{aligned} \tag{3}$$

where $\bar{\boldsymbol{A}}, \bar{\boldsymbol{B}}, \bar{\boldsymbol{C}}, \bar{\boldsymbol{D}}$ are calculated based on $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ and Δ. Initializing $\boldsymbol{A}$ by the HIPPO matrix, as proposed by (Gu et al. 2020), enhances the SSM model's ability to capture long-range contexts. Similar to previous works (Gu, Goel, and Ré 2021; Gupta, Gu, and Berant 2022), $\boldsymbol{D}$ is represented here by a skip connection. Since Eq. 3 is linear and time-invariant, the output sequence $y(t)$ can be computed directly in parallel based on input sequence $u(t)$ by convolution as follows:

$$\begin{aligned} y_k &= \bar{C}\bar{A}^k\bar{B}u_0 + \bar{C}\bar{A}^{k-1}\bar{B}u_1 + ... + \bar{C}\bar{A}\bar{B}u_{k-1} + \bar{C}\bar{B}u_k \\ y &= \bar{\boldsymbol{K}} * u \end{aligned} \tag{4}$$

Table 1: Average returns and standard deviations over 5 random seeds on the Warehouse domain.

Method	Tiny (11x11)			Small (11x20)		
	(N = 2)	(N = 4)	(N = 6)	(N = 2)	(N = 4)	(N = 6)
BC	8.80 ± 0.25	11.12 ± 0.19	14.06 ± 0.32	5.54 ± 0.06	7.88 ± 0.14	8.90 ± 0.13
ICQ	9.38 ± 0.75	12.13 ± 0.44	14.59 ± 0.16	5.43 ± 0.19	7.93 ± 0.19	8.87 ± 0.22
OMAR	6.77 ± 0.64	14.39 ± 0.91	16.13 ± 1.21	4.40 ± 0.34	7.12 ± 0.38	8.41 ± 0.49
MADTKD	6.24 ± 0.60	9.90 ± 0.21	13.06 ± 0.19	3.65 ± 0.34	6.85 ± 0.36	7.85 ± 0.52
OptiDICE	8.70 ± 0.06	11.13 ± 0.44	14.02 ± 0.36	4.84 ± 0.32	7.68 ± 0.09	8.47 ± 0.26
AlberDICE	**11.15 ± 0.35**	13.11 ± 0.32	15.72 ± 0.36	5.97 ± 0.11	8.18 ± 0.19	9.65 ± 0.13
MADS4 (ours)	**11.79 ± 0.61**	**15.52 ± 0.20**	**17.29 ± 0.76**	**6.58 ± 0.28**	**9.47 ± 0.15**	**10.87 ± 0.55**

Table 2: Average returns and standard deviations over 5 random seeds on the SMAC domain.

		RL-based			Sequence-based	
SMAC Map	Data	ICQ	OMAR	OMIGA	MADT	**MADS4 (ours)**
5m vs 6m (H)	G	7.87 ± 0.30	7.40 ± 0.63	**8.25 ± 0.37**	8.15 ± 0.63	8.00 ± 0.45
	M	7.77 ± 0.30	7.08 ± 0.51	**7.92 ± 0.57**	7.80 ± 0.56	7.85 ± 0.57
	P	7.26 ± 0.19	7.27 ± 0.42	7.52 ± 0.21	7.23 ± 0.48	**7.67 ± 0.15**
2c vs 64zg (H)	G	18.82 ± 0.17	17.27 ± 0.78	19.15 ± 0.32	18.90 ± 0.78	**19.40 ± 0.55**
	M	15.57 ± 0.61	10.20 ± 0.20	16.03 ± 0.19	16.92 ± 0.20	**17.27 ± 0.15**
	P	12.56 ± 0.18	11.33 ± 0.50	13.02 ± 0.66	13.33 ± 0.50	**14.67 ± 0.32**
6h vs 8z (SH)	G	11.81 ± 0.12	9.85 ± 0.28	12.54 ± 0.21	12.55 ± 0.67	**12.75 ± 0.15**
	M	11.13 ± 0.33	10.36 ± 0.16	12.31 ± 0.22	12.36 ± 0.16	**12.57 ± 0.25**
	P	10.55 ± 0.10	10.63 ± 0.25	11.67 ± 0.19	11.63 ± 0.25	**11.89 ± 0.43**
corridor (SH)	G	15.54 ± 1.12	6.74 ± 0.69	15.88 ± 0.89	**17.81 ± 1.14**	16.02 ± 0.97
	M	11.30 ± 1.57	7.26 ± 0.71	11.66 ± 1.30	12.75 ± 1.18	**12.80 ± 1.12**
	P	4.47 ± 0.33	4.28 ± 0.49	5.61 ± 0.35	**8.76 ± 0.49**	8.57 ± 0.54

where $\bar{\boldsymbol{K}}$ is the SSM convolution kernel which is a function of $\bar{\boldsymbol{A}}, \bar{\boldsymbol{B}}, \bar{\boldsymbol{C}}, \bar{\boldsymbol{D}}$ and a fixed context length L during training. This non-circular convolution enables parallelizable training across time steps, while the recurrent view allows fast inference with constant memory. These features make SSMs better suited than transformers for reinforcement learning, where efficient online interaction is crucial. More detailed comparisons of computational complexity and parameter efficiency are provided in the Appendix.

Information sharing with minimal communication To enable scalable cooperation among S4-based agents, we design a communication mechanism limited to consecutive agents in the SE-MDP sequence. Each agent's memory, represented by the hidden state of its S4 module, encodes information about all prior agents in the sequence. A projection of this latent state, h_{i-1}^t, is passed as input to the next agent along with other inputs $\hat{u}_i^t$, influencing its action a_i^t and its memory h_i^t:

$$a_i^t, h_i^t = \pi_i(\hat{u}_i^t, h_{i-1}^t; \theta_i) \tag{5}$$

During training, gradients flow backward through the shared latent states, enabling the entire system to learn cooperative strategies:

$$\frac{\partial J}{\partial \theta_i} = \frac{\partial J}{\partial a_i^t} \cdot \frac{\partial a_i^t}{\partial \theta_i} + \frac{\partial J}{\partial a_{i+1}^t} \cdot \frac{\partial a_{i+1}^t}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial \theta_i}. \tag{6}$$

where J represents the supervised loss function computed across all agents in the system. This sequential flow of information eliminates the need for an agent to communicate with more than one peer or identify useful collaborators, a challenge that grows with the number of agents. In contrast to typical communication-based MARL algorithms, which scale poorly with the number of agents, our mechanism is highly efficient, requiring only constant memory per agent.

MADS4: Offline Training with Online Finetuning

We adapt Decision S4 (DS4) for each agent with shared parameters, similar to Multi-Agent Decision Transformer (MADT) (Meng et al. 2021). However, unlike MADT, MADS4 trains agents sequentially, enabling memory sharing between consecutive agents. Offline training is performed using pre-collected trajectories, followed by on-policy fine-tuning of pre-trained models utilizing MAPPO (Yu et al. 2021).

In the offline training of MADS4, each agent is trained on trajectories containing sequences of prior observations, executed actions, the latent state of its predecessor, and returns-to-go from the current timestep. Similar to MADT, the state of each agent at each time step s_i^t is composed of global environment state s_{gi}^t and its local observation o_i^t. Thus, a trajectory for the i^{th} agent, consists of the following:

$$\tau_i = (u^1, u^2, ..., u^T) \quad \text{where} \quad u^t = \{R^t, s_{gi}^t, o_i^t, a_i^{t-1}, h_{i-1}^t\} \tag{7}$$

where R^t is the returns-to-go, s_{gi}^t the global state, o_i^t the local observation, a_i^{t-1} the agent's previous action, and h_{i-1}^t the hidden state of its predecessor. The S4-based model predicts actions autoregressively:

$$\hat{a}_i^t = \arg\max_a P(a_i^t | \tau_i^{<=t}; \theta) \tag{8}$$

where θ represents shared model parameters for all agents, ensuring training stability. Each agent's unique inputs and one-hot IDs are used to account for agent-specific behavior. More details on network architecture and offline training are provided in the Appendix.

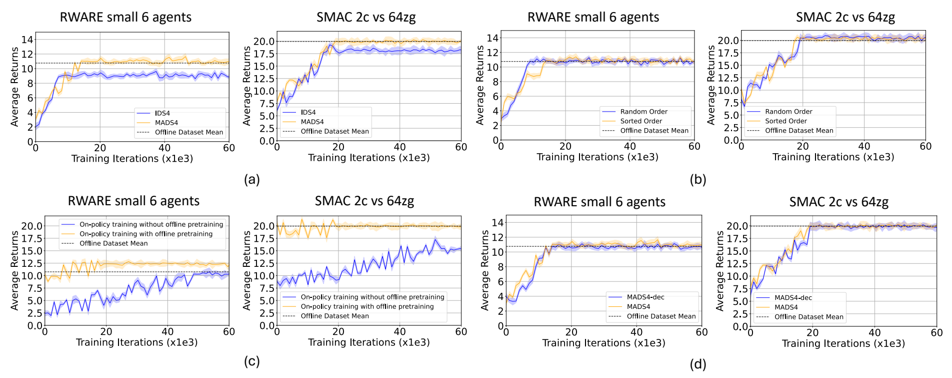

Figure 2: (a) Comparison of MADS4 with information sharing between agents and IDS4 where agents are trained independently. (b) Comparison of random order against sorted order of the agents. (c) Effect of on-policy fine-tuning with pre-training vs. without pretraining. (d) Comparison of MADS4 against decentralized MADS4. Mean and standard deviations of average returns are plotted over 5 independent runs.

Experiments

Datasets and Baselines We evaluate MADS4 on challenging cooperative MARL benchmarks: Multi-Robot Warehouse (RWARE) (Papoudakis et al. 2020) and StarCraft2 Multi-Agent Challenge (SMAC) (Samvelyan et al. 2019). RWARE datasets, sourced from (Matsunaga et al. 2023), include expert trajectories from Multi-Agent Transformer (MAT) (Wen et al. 2022). For SMAC, we use datasets from (Meng et al. 2021), comprising trajectories from MAPPO agents of good, medium, and poor quality, tested on two hard and two super hard maps. We compare MADS4 against recent offline MARL algorithms from both offline reinforcement learning and sequence-based supervised learning paradigms. Additional details on datasets and baselines are in the Appendix.

Offline Training Tables 1 and 2 show the mean and standard deviation of average returns in the RWARE and SMAC domains, evaluated over 30 episodes and 5 training seeds. During evaluation, the desired returns-to-go is set at 10% higher than the highest returns encountered in the offline datasets. In the **RWARE** domain, MADS4 outperforms all baselines across the maps, with a larger performance gap on the small and tiny maps involving 6 agents, where tight coordination is crucial to avoid collisions in confined spaces. MADS4 also outperforms transformer-based baselines like MADTKD, likely due to the long trajectories in the RWARE datasets (up to 500 timesteps), which are often truncated to reduce transformer training costs. In contrast, MADS4 processes full trajectories, capturing longer contexts with fewer parameters. In the **SMAC** domain, MADS4 consistently matches or exceeds the performance of all baselines across the considered hard and superhard maps, particularly in the 2c vs. 64zg and 6h vs. 8z scenarios.

Effect of sharing information Sharing information between agents significantly enhances cooperation, yielding higher rewards (Figure 2(a)), especially in complex tasks requiring precise coordination. The scalable method involves minimal overhead by limiting communication to neighboring agents, using action logits, latent states, or both. Appendix analyzes the impact of these information types.

Effect of order of agents We evaluated the impact of agent ordering on MADS4's performance by comparing two settings: (1) Random Order (agents shuffled during training) and (2) Sorted Order (dataset order). Figure 2(b) shows similar performance, demonstrating MADS4's robustness to agent ordering in the SE-MDP framework.

On-policy fine-tuning On-policy fine-tuning of the offline pre-trained MADS4 model, shown in Figure 2(c), is shown to improve performance. Without pre-training, on-policy training leads to sub-optimal performance across all tasks.

Decentralization of MADS4 To adapt MADS4 for a decentralized setting, where agents act in parallel, we leverage the hidden state information of each agent from the previous timestep as a proxy for the current timestep. This approach removes the sequential dependency in updating agent memory, as all agents' memory information from the previous timestep is available when making decisions at the current timestep. Since memory accumulates over multiple timesteps, relying on the previous timestep's information does not compromise performance, as demonstrated in Figure 2(d).

Conclusions and Discussions

In this work, we showcase the effectiveness of S4-based models in surpassing transformer-based architectures for sequence-to-sequence offline multi-agent reinforcement learning (MARL) tasks. By restricting communication to the exchange of information between unique, arbitrarily selected pairs of agents, MADS4 fosters superior cooperation compared to state-of-the-art offline RL and centralized transformer-based baselines, which require complete access to all agents' information during training. MADS4 offers a low-latency and lightweight model that can be trained more efficiently than transformers and fine-tuned online using recurrent computations.

References

Bai, C.; Wang, L.; Yang, Z.; Deng, Z.; Garg, A.; Liu, P.; and Wang, Z. 2022. Pessimistic bootstrapping for uncertainty-driven offline reinforcement learning. *arXiv preprint arXiv:2202.11566*.

Bar-David, S.; Zimerman, I.; Nachmani, E.; and Wolf, L. 2023. Decision s4: Efficient sequence-based rl via state spaces layers. *arXiv preprint arXiv:2306.05167*.

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Cao, Y.; Yu, W.; Ren, W.; and Chen, G. 2012. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial informatics*, 9(1): 427–438.

Chen, L.; Lu, K.; Rajeswaran, A.; Lee, K.; Grover, A.; Laskin, M.; Abbeel, P.; Srinivas, A.; and Mordatch, I. 2021. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34: 15084–15097.

Cheng, P.; Zhan, X.; Zhang, W.; Lin, Y.; Wang, H.; Jiang, L.; et al. 2024. Look beneath the surface: Exploiting fundamental symmetry for sample-efficient offline rl. *Advances in Neural Information Processing Systems*, 36.

Fujimoto, S.; Meger, D.; and Precup, D. 2019. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, 2052–2062. PMLR.

Gu, A.; Dao, T.; Ermon, S.; Rudra, A.; and Ré, C. 2020. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487.

Gu, A.; Goel, K.; and Ré, C. 2021. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*.

Gu, A.; Johnson, I.; Goel, K.; Saab, K.; Dao, T.; Rudra, A.; and Ré, C. 2021. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 34: 572–585.

Gupta, A.; Gu, A.; and Berant, J. 2022. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems*, 35: 22982–22994.

Jiang, J.; and Lu, Z. 2023. Offline Decentralized Multi-Agent Reinforcement Learning. In *ECAI*, 1148–1155.

Kostrikov, I.; Fergus, R.; Tompson, J.; and Nachum, O. 2021. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*, 5774–5783. PMLR.

Kumar, A.; Fu, J.; Soh, M.; Tucker, G.; and Levine, S. 2019. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in neural information processing systems*, 32.

Kumar, A.; Zhou, A.; Tucker, G.; and Levine, S. 2020. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 1179–1191.

Lee, J.; Jeon, W.; Lee, B.; Pineau, J.; and Kim, K.-E. 2021. Optidice: Offline policy optimization via stationary distribution correction estimation. In *International Conference on Machine Learning*, 6120–6130. PMLR.

Leroy, P.; Morato, P. G.; Pisane, J.; Kolios, A.; and Ernst, D. 2023. IMP-MARL: a Suite of Environments for Large-scale Infrastructure Management Planning via MARL. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Levine, S.; Kumar, A.; Tucker, G.; and Fu, J. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Li, C.; Liu, J.; Zhang, Y.; Wei, Y.; Niu, Y.; Yang, Y.; Liu, Y.; and Ouyang, W. 2023. Ace: Cooperative multi-agent q-learning with bidirectional action-dependency. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, 8536–8544.

Li, J.; Zhan, X.; Xu, H.; Zhu, X.; Liu, J.; and Zhang, Y.-Q. 2022. When data geometry meets deep function: Generalizing offline reinforcement learning. *arXiv preprint arXiv:2205.11027*.

Lu, C.; Schroecker, Y.; Gu, A.; Parisotto, E.; Foerster, J.; Singh, S.; and Behbahani, F. 2024. Structured state space models for in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Matsunaga, D. E.; Lee, J.; Yoon, J.; Leonardos, S.; Abbeel, P.; and Kim, K.-E. 2023. AlberDICE: addressing out-of-distribution joint actions in offline multi-agent RL via alternating stationary distribution correction estimation. *Advances in Neural Information Processing Systems*, 36: 72648–72678.

Meng, L.; Wen, M.; Yang, Y.; Le, C.; Li, X.; Zhang, W.; Wen, Y.; Zhang, H.; Wang, J.; and Xu, B. 2021. Offline pre-trained multi-agent decision transformer: One big sequence model tackles all smac tasks. *arXiv preprint arXiv:2112.02845*.

Pan, L.; Huang, L.; Ma, T.; and Xu, H. 2022. Plan better amid conservatism: Offline multi-agent reinforcement learning with actor rectification. In *International conference on machine learning*, 17221–17237. PMLR.

Papoudakis, G.; Christianos, F.; Schäfer, L.; and Albrecht, S. V. 2020. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. *arXiv preprint arXiv:2006.07869*.

Samvelyan, M.; Rashid, T.; De Witt, C. S.; Farquhar, G.; Nardelli, N.; Rudner, T. G.; Hung, C.-M.; Torr, P. H.; Foerster, J.; and Whiteson, S. 2019. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*.

Shao, J.; Qu, Y.; Chen, C.; Zhang, H.; and Ji, X. 2023. Counterfactual Conservative Q Learning for Offline Multi-agent Reinforcement Learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Smith, J. T.; Warrington, A.; and Linderman, S. W. 2022. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*.

Tseng, W.-C.; Wang, T.-H. J.; Lin, Y.-C.; and Isola, P. 2022. Offline multi-agent reinforcement learning with knowledge distillation. *Advances in Neural Information Processing Systems*, 35: 226–237.

Tustin, A. 1947. A method of analysing the behaviour of linear systems in terms of time series. *Journal of the Institution of Electrical Engineers-Part IIA: Automatic Regulators and Servo Mechanisms*, 94(1): 130–142.

Uchendu, I.; Xiao, T.; Lu, Y.; Zhu, B.; Yan, M.; Simon, J.; Bennice, M.; Fu, C.; Ma, C.; Jiao, J.; et al. 2023. Jump-start reinforcement learning. In *International Conference on Machine Learning*, 34556–34583. PMLR.

Vinitsky, E.; Lichtle, N.; Parvate, K.; and Bayen, A. 2020. Optimizing mixed autonomy traffic flow with decentralized autonomous vehicles and multi-agent rl. *arXiv preprint arXiv:2011.00120*.

Wang, X.; Xu, H.; Zheng, Y.; and Zhan, X. 2024. Offline multi-agent reinforcement learning with implicit global-to-local value regularization. *Advances in Neural Information Processing Systems*, 36.

Wen, M.; Kuba, J.; Lin, R.; Zhang, W.; Wen, Y.; Wang, J.; and Yang, Y. 2022. Multi-agent reinforcement learning is a sequence modeling problem. *Advances in Neural Information Processing Systems*, 35: 16509–16521.

Wu, Y.; Tucker, G.; and Nachum, O. 2019. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*.

Wu, Y.; Zhai, S.; Srivastava, N.; Susskind, J.; Zhang, J.; Salakhutdinov, R.; and Goh, H. 2021. Uncertainty weighted actor-critic for offline reinforcement learning. *arXiv preprint arXiv:2105.08140*.

Xu, H.; Jiang, L.; Jianxiong, L.; and Zhan, X. 2022. A policy-guided imitation approach for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 35: 4085–4098.

Xu, H.; Jiang, L.; Li, J.; Yang, Z.; Wang, Z.; Chan, V. W. K.; and Zhan, X. 2023. Offline rl with no ood actions: In-sample learning via implicit value regularization. *arXiv preprint arXiv:2303.15810*.

Xu, H.; Zhan, X.; Li, J.; and Yin, H. 2021. Offline reinforcement learning with soft behavior regularization. *arXiv preprint arXiv:2110.07395*.

Xu, H.; Zhan, X.; and Zhu, X. 2022. Constraints penalized q-learning for safe offline reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 8753–8760.

Yang, Y.; Ma, X.; Li, C.; Zheng, Z.; Zhang, Q.; Huang, G.; Yang, J.; and Zhao, Q. 2021. Believe what you see: Implicit constraint approach for offline multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 10299–10312.

Ye, D.; Zhang, M.; and Yang, Y. 2015. A multi-agent framework for packet routing in wireless sensor networks. *sensors*, 15(5): 10026–10047.

Yu, C.; Velu, A.; Vinitsky, E.; Wang, Y.; Bayen, A.; and Wu, Y. 2021. The surprising effectiveness of PPO in cooperative, multi-agent games (2021). *arXiv preprint arXiv:2103.01955*.

Yu, T.; Thomas, G.; Yu, L.; Ermon, S.; Zou, J. Y.; Levine, S.; Finn, C.; and Ma, T. 2020. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33: 14129–14142.

Additional Background and Related Work

Offline Reinforcement Learning Offline RL allows for policy learning based on pre-collected datasets without having access to active interactions with the environment (Levine et al. 2020), which is then directly used as the final policy or is used as a starting point for further improvement (Uchendu et al. 2023). This learning paradigm, however, results in severe distribution shift and extrapolation errors during policy evaluation on OOD samples not present in the offline dataset (Kumar et al. 2019; Fujimoto, Meger, and Precup 2019). Several approaches have been developed to mitigate this issue which typically involves various types of regularizations to be near the offline data distribution. Policy-based regularizations implicitly or explicitly constrain the policy to be close to the behavior policy of the dataset (Wu, Tucker, and Nachum 2019; Xu et al. 2021; Cheng et al. 2024; Li et al. 2022). Value-based regularizations aim to learn conservative value functions on OOD samples (Kumar et al. 2020; Kostrikov et al. 2021; Xu, Zhan, and Zhu 2022). Other approaches involve including uncertainty (Wu et al. 2021; Bai et al. 2022) or penalizing OOD rewards (Yu et al. 2020).

On the other hand, Decision Transformer (DT) (Chen et al. 2021) takes a goal-conditioned supervised learning (GCSL) approach to formulate offline RL as a sequence modeling task and outperforms many state-of-the-art offline RL algorithms. Following the success of this training regime, Decision S4 (Bar-David et al. 2023) proposes using S4 model variants for higher parameter efficiency, capturing longer sequences and faster inference.

Offline MARL Extending single-agent offline RL methods (Levine et al. 2020; Kumar et al. 2019; Fujimoto, Meger, and Precup 2019; Wu, Tucker, and Nachum 2019; Xu et al. 2021; Cheng et al. 2024; Li et al. 2022; Kumar et al. 2020; Kostrikov et al. 2021; Xu, Zhan, and Zhu 2022; Bai et al. 2022) to multi-agent settings presents significant challenges due to the exponential growth of the joint state-action space, and global-level policy regularization are challenging to obtain and may result in very sparse constraints, especially with limited and less diverse offline dataset. Therefore, recently, offline RL-based MARL algorithms have emerged, typically applying regularizations on local policies or value functions (Yang et al. 2021; Jiang and Lu 2023; Pan et al. 2022). On the other hand, (Meng et al. 2021) extends the sequence-learning based Decision Transformer (DT) (Chen et al. 2021) to a multi-agent setting, where agents are trained independently by sharing weights within a goal-conditioned supervised learning framework. However, these algorithms do not provide guarantees of global-level regularizations and fail to explicitly or implicitly learn cooperative behavior. Only a few recent works have tried to tackle these limitations. For example, (Wang et al. 2024) uses an implicit global to-local regularization, and (Tseng et al. 2022) uses knowledge distillation to distill cooperation in the local policies.

S4 S4(Gu, Goel, and Ré 2021; Gu et al. 2021) and their variants (Gupta, Gu, and Berant 2022; Smith, Warrington, and Linderman 2022), which are developed on time-invariant linear state space layers, have outperformed transformers in capturing long-range contexts. These models require far fewer parameters and have constant time inference; hence, they have been suitably utilized in reinforcement learning domains in single-agent learning (Bar-David et al. 2023) and in-context learning (Lu et al. 2024). Commonly, the model uses the convolutional mode for efficient parallelizable training (where the whole input sequence is seen ahead of time) and switched into a recurrent mode for efficient autoregressive inference (where the inputs are seen one timestep at a time).

Additional Details on Experimental Setup and Training Details

Sequence-based reinforcement learning

Offline RL is formulated as a supervised learning problem by predicting actions in an autoregressive manner typically conditioned on current states, previously executed actions, and desired returns to go. This paradigm was introduced in (Chen et al. 2021), where the DT is trained to predict current actions based on returns to go instead of current rewards in order to have better actions that are correlated with better future rewards. This work utilizes this supervised learning setting, where the state, action, and reward of i^{th} agent at each timestep are denoted as s_i, a_i, r_i, and its trajectories $\tau : (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_L, a_L, r_L)$ consist of sequences of state, action, and reward tuples. Since the models are trained on returns-to-go, the trajectories are restructured as $\tau : (R_0, s_0, a_0, R_1, s_1, a_1, ..., R_L, s_L, a_L)$ where $R_i = \sum_{t=i}^{N} r_i$ is the returns to go from i^{th} time step.

Network Architecture and Offline Training The MADS4 architecture comprises three main components (Figure 1): (i) Projection layers, which consist of fully connected layers followed by ReLU activations; (ii) Input encoder layers, which handle states, actions, and rewards as fully connected linear layers; (iii) Sequence modeling, which features stacked S4 blocks with Batch Normalization, S4 layers, linear mixing with GELU activation, and dropout.

The "Normal Plus Low Rank" kernel, initialized with HIPPO, provides the best performance. For all experiments, the input channel and S4 state sizes are set to $H = 96$ and $N = 96$. An ablation study on state and input sizes is in the Appendix.

In the offline setting, the S4 model is trained efficiently using the convolutional view on entire trajectories sampled randomly from the offline dataset. The trajectories are zero-padded to a constant context length. Unlike transformers, which face limitations on context length due to the expensive quadratic time and space complexity of self-attention, S4-based models can be trained on complete trajectories that are often much longer than those typically used for transformers in most environments. The impact of truncating trajectory lengths has significant implications for model performance, as detailed later in the Appendix. Actions are predicted based on the action logit outputs of the model, and the model is trained based on loss computed using cross-entropy between the true action labels and the predicted actions.

Network Architecture and Online Finetuning The pre-trained model is loaded as the actor network, which predicts action probabilities and next states as:

$$p_i^t, h_i^t = \pi(u_i^t, h_i^{t-1}; \theta) \quad \text{where} \quad u_i^t = \{R_i^t, s_{gi}^t, o_i^t, a_i^{t-1}, h_{i-1}^t\} \tag{9}$$

The critic network is parameterized by fully connected layers with ReLU activations, which take the shared global state of the environment and encoded latent states and evaluate the value function, which is used to update the S4-based actor parameters (θ) using the policy gradient theorem. For more stable training. the actor network is kept frozen initially, and the critic is solely trained on the recorded data collected using the pre-trained actor. After sufficient training of the critic, the actor and critics are simultaneously trained. During exploration, the desired returns-to-go is set at 10% higher than the current model's highest return.

Other Training Details

In all experiments, we set the input channel size to $H = 96$ and the S4 state size to $N = 96$. Offline training is conducted on batches of 64 trajectories, with the maximum trajectory length in the offline dataset used as the length for each batch. The shorter trajectories are zero-padded to a constant length. The training was performed using Adam optimizer with a learning rate of 10^{-4}.

The offline trained model is fine-tuned online using on-policy MAPPO. During the initial stage of fine-tuning, the actor network is kept frozen, and the critic is first trained for the first 50,000 iterations. After this, both the actor and critic are trained simultaneously, with a slower learning rate for the actor network (10^{-5}) compared to the critic (10^{-4}). During on-policy fine-tuning, the returns-to-go is set at 10% higher than the highest returns encountered during training. On-policy training is conducted in batches of 64. To mitigate the issue of deteriorating performance with prolonged on-policy training, the S4 kernel A can be kept frozen. All experiments were run on a single NVIDIA RTX 2080Ti GPU. Experiments on the RWARE domain take less than 2 hrs to reach optimal performance, and experiments on the SMAC domain take less than 6hrs, 12 hrs, 12 hrs, and 30 hrs for maps 2c vs. 64zg, 5m vs. 6m, 6h vs. 8z and Corridor, respectively.

S4 Layer

S4(Gu, Goel, and Ré 2021) layer is a variant of linear and time-invariant (LTI) state-space model (SSM)(Gu et al.

2021) which adopts the HIPPO (Gu et al. 2020)-based initializations in order to better capture longer contexts, and proposes efficient ways for kernel computations and parallel training.

Recurrent View Given an input scalar function $u(t) : \mathbb{R} \to \mathbb{R}$, the continuous LTI SSM is defined by the following first-order differential equation:

$$\dot{x}(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t) + Du(t) \tag{10}$$

The model maps the input stream $u(t)$ to $y(t)$. It was shown that initializing A by the HIPPO matrix (Gu et al. 2020) grants the state-space model (SSM) the ability to capture long-range dependencies. Similar to previous works (Gu, Goel, and Ré 2021; Gupta, Gu, and Berant 2022), D is replaced by parameter-based skip-connection and is omitted from the SSM by assuming $D = 0$.

This SSM operates on continuous sequences, and it is discretized by a step size Δ to operate on discrete sequences. Let the discretization matrices be $\bar{A}, \bar{B}, \bar{C}$:

$$\begin{aligned} \bar{A} &= (I - \Delta A/2)^{-1}(I + \Delta A/2), \\ \bar{B} &= (I - \Delta A/2)^{-1}\Delta B, \\ \bar{C} &= C. \end{aligned} \tag{11}$$

These matrices allow us to rewrite Eq. 10:

$$x_k = \bar{A}x_{k-1} + \bar{B}u_k, \quad y_k = \bar{C}x_k \tag{12}$$

Using the recurrent Eq.12, SSM asymptotically allows for constant $O(1)$ time and memory inference for each token/ timestep, as compared to $O(L^2)$ inference for transformers. SSM can be interpreted as a linear RNN in which $\bar{A}$ is the state-transition matrix, and $\bar{B}, \bar{C}$ are the input and output matrices. Thus, it essentially requires $O(L)$ training, L being the sequence length, as compared to $O(L^2)$ (parallelizable) training complexity for transformers.

Convolutional View The recurrent SSM view is not practical for training over long sequences, as the training cannot be parallelized across the sequence dimension and results in instabilities from vanishing gradient issues. However, the LTI SSM can be rewritten as a convolution, which allows for efficient parallelizable training. The S4 convolutional view is obtained as follows:

Given a sequence of scalars $u = (u_0, u_1, ..., u_{L-1})$ of length L, the S4 recurrent view can be unrolled to the following closed form:

$$\begin{aligned} \forall i \in [L-1] : \quad & x_i \in \mathbb{R}^N, \\ & x_0 = \bar{B}u_0, \\ & x_1 = \bar{A}\bar{B}u_0 + \bar{B}u_1, \\ & \ldots, \\ & x_{L-1} = \sum_{i=0}^{L-1} \bar{A}^{L-1-i}\bar{B}u_i. \end{aligned} \tag{13}$$

$$\begin{aligned} & y_i \in \mathbb{R}, \\ & y_0 = \bar{C}\bar{B}u_0, \\ & y_1 = \bar{C}\bar{A}\bar{B}u_0 + \bar{C}\bar{B}u_1, \\ & \ldots, \\ & y_{L-1} = \sum_{i=0}^{L-1} \bar{C}\bar{A}^{L-1-i}\bar{B}u_i. \end{aligned} \tag{14}$$

where N is the state size. Inputs and outputs are scalars. Since the recurrent rule is linear, it can be computed in closed form with matrix multiplication or non-circular convolution:

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{L-1} \end{bmatrix} = \begin{bmatrix} \bar{C}\bar{B} & 0 & 0 & \cdots & 0 \\ \bar{C}\bar{A}\bar{B} & \bar{C}\bar{B} & 0 & \cdots & 0 \\ \bar{C}\bar{A}^2\bar{B} & \bar{C}\bar{A}\bar{B} & \bar{C}\bar{B} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \bar{C}\bar{A}^{L-1}\bar{B} & \bar{C}\bar{A}^{L-2}\bar{B} & \bar{C}\bar{A}^{L-3}\bar{B} & \cdots & \bar{C}\bar{B} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{L-1} \end{bmatrix}$$

i.e., $y = \bar{k} * u$ for some kernel $\bar{k}$, which can be calculated by fixing the sequence length L before training. This kernel can be efficiently computed using FFT operations; for example, (Gu, Goel, and Ré 2021) computes the kernel via inverse FFT on the spectrum of $\bar{k}$, which is calculated via Cauchy kernel and the Woodbury Identity. This benefits from the "Normal Plus Low Rank" parameterization of the HIPPO-initialized state transition matrix A, and other more efficient parameterizations are proposed in (Gupta, Gu, and Berant 2022).

The SSM, as represented above, operates on scalars or one channel of inputs. To handle vector inputs $\in \mathbb{R}^H$, H copies of the 1-D SSM layer are stacked, one for each input channel, and a linear mixing layer in the after block of the S4 layer mixes the information from different channels to produce outputs $\in \mathbb{R}^H$.

Sharing Hidden State Representations

The raw outputs from the S4 layer consist of $y_k = \bar{C}x_k$, where $y_k \in \mathbb{R}^H$ and the latent states $x_k \in \mathbb{R}^{N \times H}$ for H input channels. Since the outputs are linear projections and offer a compact representation of the latent states (or, memory of the agent), this has been used as the message that is transmitted from one agent to the next in the SE-MDP. This offers several advantages: i) results in better team performance; ii) offers scalable cooperation between agents, which eliminates the need for a centralized transformer or a critic, which requires access to information from all agents; one agent needs access to only its immediate neighbor in the sequence; (iii) allows parallel training via convolution.

We also experimented with passing the raw hidden states $x_k \in \mathbb{R}^{N \times H}$ from one agent to another. The hidden states can be complex, depending on the parameterization of the S4 kernel. Therefore, before passing the latent states directly, we first linearly mix the hidden states across the H channels to obtain $x_k \in \mathbb{C}^N$. Then, we linearly project the real and imaginary parts of x_k after concatenation. This

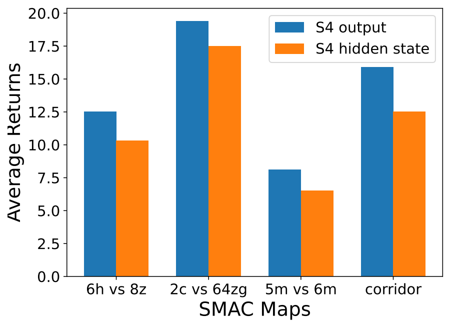

Figure 3: Average Returns obtained in SMAC tasks by passing S4 output versus S4 latent states.

mode of information transfer, however, has notable drawbacks: i) it requires computing the S4 hidden state at every timestep, which requires recurrent rollouts of the S4 kernel, and ii) it fails to outperform the method of passing the S4 outputs; possibly due to errors accumulated during recurrent training. A comparison of performance using S4 output representation versus S4 latent state representation is shown in Figure3, where passing S4 outputs resulted in better performance across all tasks.

It is, however, noted that hidden states at each timestep may be efficiently obtained utilizing the parallel (associate) scan operation as done in (Smith, Warrington, and Linderman 2022; Lu et al. 2024), but this requires JAX implementation and is currently not supported by PyTorch.

Preliminary study using Mamba

We also explored Mamba as an alternative to LTI S4-based models. Mamba allows time-variant parameters to be considered in the SSM equations. Though convolution cannot be applied here since the kernel cannot be computed apriori since the parameters B, C are input-dependent, efficient parallel scan operation allows for parallelizable $O(\log L)$ complexity. However, preliminary analysis utilizing Mamba resulted in suboptimal performance, and it requires more extensive analysis.

Datasets and Baselines

RWARE

RWARE environment is a warehouse simulation consisting of agents moving and delivering goods to workstations in partially observable settings while avoiding collisions. This domain poses challenges due to high-dimensional observations and the need for strong cooperation, especially in high-density settings where agents must navigate narrow passages. The offline dataset on RWARE (Papoudakis et al. 2020) is obtained from (Matsunaga et al. 2023), which contains an expert dataset with diverse behaviors obtained by training MAT on small and tiny maps. The dataset consists of 1000 trajectories, each trajectory consisting of 500 timesteps. The dataset statistics are in Table 3. The longest trajectories consist of timesteps in the range of 500 in all the datasets.

The baseline results are obtained from (Matsunaga et al. 2023), which currently holds the state-of-the-art results of the baselines listed on this dataset.

Map Name	Maximum	Minimum	Average
small 2 agents	12.37	1.13	7.12
small 4 agents	12.08	3.93	9.49
small 6 agents	12.69	7.59	10.76
tiny 2 agents	16.81	1.97	12.77
tiny 4 agents	18.63	10.40	15.67
tiny 6 agents	19.97	11.88	17.45

Table 3: RWARE datasets

SMAC

The offline SMAC (Samvelyan et al. 2019) dataset is obtained from (Wang et al. 2024). This dataset is obtained by randomly sampling 1000 trajectories from the original dataset provided by (Meng et al. 2021). We consider 4 representative battle maps, including 2 hard maps (5m vs 6m, 2c vs 64zg) and 2 super hard maps (6h vs 8z, corridor), which are detailed in Table 4. The average returns for the dataset are listed in Table 5. The longest trajectories are encountered in the Corridor map, which typically comprises about 100 timesteps.

Map Name	Ally Units	Enemy Units	Type
5m_vs_6m	5 Marines	6 Marines	homogeneous & asymmetric
2c_vs_64zg	2 Colossi	64 Zerglings	micro-trick: positioning
6h_vs_8z	6 Hydralisks	8 Zealots	micro-trick: focus fire
corridor	6 Zealots	24 Zerglings	micro-trick: wall off

Table 4: SMAC maps for experiments.

Map Name	Quality	Average Return
5m_vs_6m	good	20.00
	medium	11.03
	poor	8.50
2c_vs_64zg	good	19.94
	medium	13.00
	poor	8.89
6h_vs_8z	good	17.84
	medium	11.96
	poor	9.12
corridor	good	19.88
	medium	13.07
	poor	4.93

Table 5: SMAC datasets.

Baselines

Offline RL baselines include Behavior Cloning (BC) (Fujimoto, Meger, and Precup 2019), OptiDICE (Lee et al. 2021), AlberDICE (Matsunaga et al. 2023), ICQ (Yang et al. 2021), OMAR (Pan et al. 2022), and OMIGA (Wang et al.

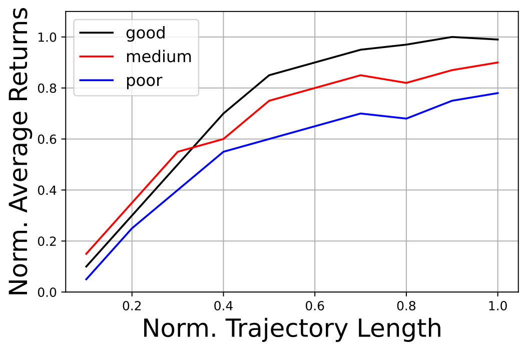

Figure 4: The effect of truncating the trajectory length during training. The average returns are normalized with the maximum returns encountered in the offline dataset.

2024). Sequence-based baselines include transformer-based MADT (Meng et al. 2021), which lacks inter-agent cooperation, and MADTKD (Tseng et al. 2022), which incorporates cooperation via a centralized teacher model. The offline RL-based baseline results are obtained from (Wang et al. 2024), and MADT results are obtained by running the code available with (Meng et al. 2021).

Hyperparameters and Additional Analysis

S4 model size parameters

We analyze the impact of the S4 model size parameters, specifically the number of input channels (H) and the latent state size (N), on the model performance, as shown in Table 6. We compare the total number of parameters against the 1.8 million parameters reported for MADTKD in (Tseng et al. 2022). Our biggest model with N=96 and H=96 was used in all our experiments, which consists of about 200k parameters.

The effect of context length

The context length used for pretraining significantly impacts performance, which is also evident for transformer-based models. In our experiments, we used the maximum trajectory lengths encountered in the offline datasets for pretraining. Representative results are shown in Figure 4, which illustrates the effects of truncating the trajectory lengths to various percentages of the maximum length in the offline dataset for the SMAC map 2c vs 64zg.

Effect of freezing A during on-policy finetuning

The degrading effect on MADS4 performance during recurrent on-policy finetuning can be mitigated by freezing the S4 kernel parameter A while updating only parameters B and C, as illustrated in Figure 5. A similar observation has also been reported in (Bar-David et al. 2023).

Effect of order of agents

To assess the impact of agent ordering on MADS4's performance, we compared two training settings: (1) Random Order, where the agent order is randomly shuffled during training, and (2) Fixed Order, where agents are trained in the same sorted order as in the offline dataset. The results in Figure 6 indicate minimal to no performance difference between the two settings, demonstrating that MADS4 is robust to agent ordering within the SE-MDP framework. Nonetheless, we recommend using a random order during training to avoid introducing potential biases into the learning process.

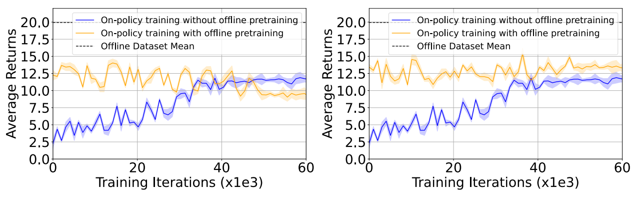

Figure 5: The effect of freezing S4 kernel parameter A in the SMAC 6h vs 8z map. Freezing A in the right Figure results in more stable performance during the on-policy recurrent finetuning.

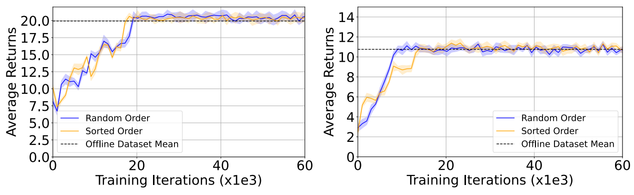

Figure 6: The effect of having a shuffled random order vs. a fixed sorted order of the agents in the SE-MDP framework on the SMAC domain in the 2c vs. 64zg map (left) and RWARE domain in the small 6 agents scenario (right).

Effect of global states as inputs

Building on prior work such as MADT, the proposed S4-based MADS4 agents utilize global states as inputs. However, in certain environments, access to the global state may be restricted or unavailable. To address this, we present an ablation study (Figure 7) evaluating the impact of using global state variables as inputs. The results indicate that omitting the global state does not lead to a significant drop in performance.

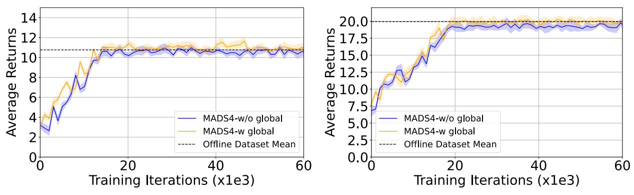

Figure 7: Performance comparison on RWARE small map with 6 agents (left) and SMAC map 2c vs 64 zg (right). The results demonstrate that excluding global states as inputs in MADS4 agents has minimal impact on performance.

Table 6: Results of smaller models on the RWARE small map. Each of the smaller models is denoted by (i) N, the $S4$ state size, and (ii) H, the number of input/output channels.

Environments	(N=96,H=96)	(N=64,H=64)	(N=32,H=32)	(N=64,H=96)	(N=96,H=64)	(N=32,H=64)	**MADTKD**
2 agents	6.58	6.21	5.53	6.53	6.25	5.87	3.65
4 agents	9.47	8.86	8.57	9.15	8.88	8.64	6.85
6 agents	10.87	10.31	9.55	10.76	9.97	9.85	7.85
% Parameters (Ours)	100	60	40	81	82	55	100
% Parameters (MADTKD)	12	7	5	8	8	6	100

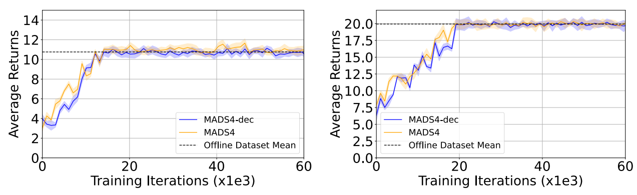

Figure 8: The comparison of performance of MADS4 vs. MADS4-dec (decentralized MADS4) on RWARE small map with 6 agents (left) and SMAC map 2c vs 64 zg (right).

MADS4 vs. Decentralized MADS4

When decisions are made at the current timestep, all decisions from the previous timestep will already be finalized. As a result, the memory information of all agents is readily available for use. This eliminates the need for any agent to wait for its peer to decide the current timestep. By utilizing the memory information from the previous timestep, agents can make decisions without relying on sequential dependencies during the current timestep. Since memory accumulates over multiple timesteps, relying on the previous timestep's information does not compromise performance, as demonstrated in Figure 8. This modification enables our algorithm to function effectively in decentralized policy settings without performance degradation.