

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

Ashmita Bhattacharya
 Pennsylvania State University
 University Park, United States
 azb6481@psu.edu

Malyaban Bal
 Pennsylvania State University
 University Park, United States
 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 MARL approaches leveraging transformers 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 achieves similar or improved performance compared to state-of-the-art offline RL and transformer-based MARL baselines across most tasks.

KEYWORDS

Multi-agent Reinforcement Learning; Offline Reinforcement Learning; State Space models

ACM Reference Format:

Ashmita Bhattacharya and Malyaban Bal. 2026. Multi-Agent Decision S4: Leveraging State Space Models for Offline Multi-Agent Reinforcement Learning. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026)*, Paphos, Cyprus, May 25 – 29, 2026, IFAAMAS, 10 pages. <https://doi.org/10.65109/YJTX3478>



This work is licensed under a Creative Commons Attribution International 4.0 License.

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaaamas.org). <https://doi.org/10.65109/YJTX3478>

1 INTRODUCTION

Multi-agent reinforcement learning (MARL) has demonstrated significant success in learning complex policies that require coordination among multiple agents to maximize a shared objective [3, 5, 48]. However, this success often relies on a substantial number of interactions with the environment, which can be computationally expensive in high-fidelity simulations or prohibitively risky in real-world applications. To enhance sample efficiency, offline reinforcement learning (RL) algorithms [9, 15–18, 21, 43, 44] have been developed, enabling learning from pre-collected offline datasets, thus reducing the need for extensive online interactions.

Offline RL is plagued by the well-known issue of distribution shift, which leads to extrapolation errors when encountering out-of-distribution (OOD) samples during policy training. This occurs when the learned policy deviates from the unknown behavior policy used to collect the training data. To mitigate this, various forms of regularization [16] are introduced to ensure that the learned policy remains close to the behavior policy [17, 43, 44]. In multi-agent settings, the joint state-action space expands exponentially as the number of agents increases. This makes it challenging to apply these regularization techniques globally on the joint state-action space, leading to sparse and less effective regularization constraints, especially when working with a limited and less diverse offline dataset.

In parallel, sequence-based supervised learning has been concurrently applied to address offline MARL, leveraging the significant success of supervised learning in capturing complex patterns from large offline datasets. This approach, first introduced by the Decision Transformer (DT) [6], has demonstrated its ability to learn policies in an autoregressive fashion by predicting the next action based on the current state, previous action and the desired return-to-go. While efforts have been made to extend DT-based architectures to offline MARL [26, 35], certain limitations persist. MADT [26] adapts DT independently for each agent in the multi-agent settings, failing to explicitly model cooperation between agents. [35] adopts a similar approach and trains a centralized teacher policy to capture agent interactions, with individual agents learning through policy distillation. However, centralized transformers face scalability issues, requiring training on information from all agents. Additionally, these approaches inherit the inherent limitations of transformers, such as large model sizes, inefficient runtime inference, and restricted ability to capture long-range dependencies due to fixed window size constraints.

Structured State Space Sequence (S4) models [11] have recently been shown to outperform transformer-based models in single-agent offline RL tasks [2]. Building on this success, we propose a sequence-learning-based offline MARL algorithm leveraging S4

variants. These models provide superior parameter efficiency compared to transformers, effectively capture longer temporal contexts, and enable constant-time inference over the quadratic time complexity of transformers with respect to the sequence length. Unlike previous works, such as 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 [20], each decision step is divided into mini-steps, with agents acting sequentially based on their predecessors’ actions. Unlike [20], we enable limited 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 previous agents during training. This design enables scalable training with constant memory communication overhead, unlike traditional communication-based MARL algorithms, where memory overhead increases quadratically with the number of agents. Additionally, this streamlined information-sharing mechanism helps mitigate non-stationarity issues during online fine-tuning. This form of training also shares similarities with the way information is passed between segments of long sequences in Recurrent Memory Transformer (RMT) [4].

The S4-based agents are trained directly on sequences or trajectories from the offline dataset in an efficient convolutional manner. The offline pre-trained models can be further used for sample-efficient online fine-tuning based on individual tuples instead of sequences leveraging the recurrent view of S4. 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) [29] and StarCraft2 Multi-Agent Challenge (SMAC) [31], where MADS4 achieves competitive or improved performance across many tasks over state-of-the-art offline RL-based and transformer-based baselines.

Our main contributions are summarized as follows:

- We introduce Multi-Agent Decision S4 (MADS4), the first offline MARL algorithm based on Structured State Space Sequence (S4) models, extending their success in single-agent offline RL to the multi-agent domain.
- MADS4 leverages an SE-MDP framework, where agents act sequentially within each timestep. Each agent accesses only its immediate predecessor’s latent S4 state, enabling scalable coordination with constant communication memory overhead. The latent state serves as a differentiable shared memory, allowing efficient information and gradient flow across agents.
- We evaluate MADS4 on challenging offline MARL benchmarks, including Multi-Robot Warehouse (RWARE) and StarCraft II Multi-Agent Challenge (SMAC), where it achieves competitive or improved performance over transformer-based and offline RL baselines.

2 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 [19], which is then directly used as the final policy or is used as a starting point for further improvement [37]. This learning paradigm however results in severe distribution shift and extrapolation errors during policy evaluation on OOD samples not present in the offline dataset [9, 16]. 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 [7, 21, 41, 45]. Value-based regularizations aim to learn conservative value functions on OOD samples [15, 17, 46]. Other approaches involve including uncertainty [1, 42] or penalizing OOD rewards [50].

In contrast to these regularization-based methods, Decision Transformer (DT) [6] 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 [2] proposes using S4 model variants for higher parameter efficiency, capturing longer sequences and faster inference.

Offline MARL. Extending single-agent RL methods to multi-agent settings presents significant challenges due to the exponential growth of the joint state-action space. Most MARL algorithms adopt the Centralized Training with Decentralized Execution (CTDE) paradigm. In CTDE, global information is shared during training, and local, decoupled policies are used for execution [8, 23, 27, 30, 33, 34, 38, 49]. Recently, offline RL-based MARL algorithms have emerged, typically applying regularizations on local policies or value functions [14, 28, 47]. On the other hand, [26] extends the Decision Transformer (DT) [6] 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, [39] uses an implicit global to-local regularization, and [35] uses knowledge distillation to distill cooperation in the local policies.

S4 Models. S4 [11, 12] and their variants [13, 32], 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 [2] and in-context learning [24]. 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).

3 METHODOLOGY

3.1 Problem Formulation

In this section, we present our cooperative multi-agent decision-making approach based on sequence learning with state-space layers. We design our approach based on the Multi-agent Markov

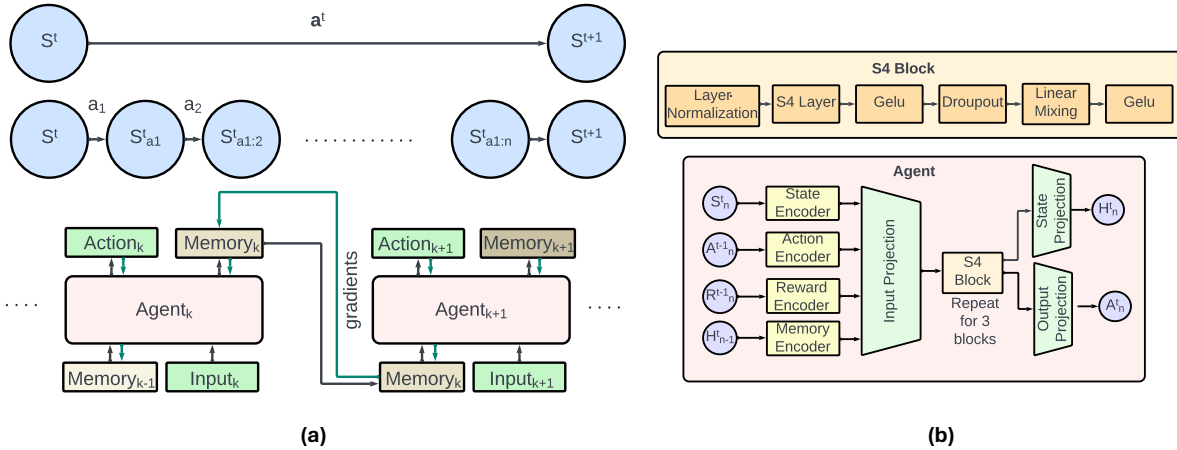


Figure 1: (a) Multi-Agent MDP (MMDP) is restructured into a Sequentially-Expanded MDP (SE-MDP) where the multi-agent state transition at each timestep is decomposed into n intermediate states. In this framework, in the forward pass (black lines), each agent processes its input alongside information received from its preceding agent, then takes an action and passes updated information to the next agent in the sequence. During training, the gradient flows backward (green lines), enabling earlier agents to receive updates based on the information passed by later agents. (b) MADRS4 Actor Network. In addition to encoding the state, action, and reward, the memory state from the previous agent (H_{n-1}) is processed through a memory encoder. The updated S4 states and outputs are then projected via separate heads.

Decision Process (MMDP) framework [22]. An MMDP over n agents is defined by the tuple $\mathcal{G} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$, where \mathcal{S} is the set of global states, and $\mathcal{A} = \mathcal{A}_1 \times \dots \times \mathcal{A}_n$ is the joint action space. At each timestep, the environment transitions from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ under the joint action $\mathbf{a} = [a_1, \dots, a_n]$, following the transition function $P(s' | s, \mathbf{a})$, and returns a global reward $r(s, \mathbf{a})$. The goal is to learn a joint policy $\pi(\mathbf{a} | s)$ that maximizes the expected return $\mathbb{E}_\pi [\sum_{t=0}^{\infty} \gamma^t r(s^t, \mathbf{a}^t)]$.

3.2 Sequentially Expanded MDP

In this work, the Multi-agent Markov Decision Process (MMDP) is transformed into a Sequentially Expanded Markov Decision Process (SE-MDP), where each timestep is divided into n mini timesteps and a multi-agent decision by n agents is expanded into a sequence of n individual decisions, with only one agent acting during each mini-timestep. Thus, a single-step transition in the original MMDP ($s^t, \mathbf{a}^t, s^{t+1}$) resulting in a shared reward $r(s^t, \mathbf{a}^t)$ is composed of a sequence of n intermediate transitions, which in turn result in the same shared reward $r(s^t, \mathbf{a}^t)$, as shown in Figure 1(a).

$$(s^t, \mathbf{a}^t, s^{t+1}) = \{(s^t, a_1^t, s_{a_1}^t), (s_{a_1}^t, a_2^t, s_{a_1:2}^t), \dots, (s_{a_1:n-1}^t, a_n^t, s_{a_1:n}^t = s^{t+1})\} \quad (1)$$

Within this framework, at each timestep, an agent’s action is based on information passed by the immediately preceding agent in the sequence. This creates a bidirectional dependency between the agents as shown in Figure 1(a): in the forward direction, an agent’s action is influenced by the actions of its predecessors, while in the backward direction, gradients can propagate from the current agent back to the previous agents.

3.3 Sequence-based Reinforcement Learning

Our approach follows the offline RL paradigm using sequence modeling, where the RL problem is reframed as a supervised learning task by predicting actions in an autoregressive manner—typically conditioned on the current state, past actions, and a target return-to-go. The return-to-go at timestep t , denoted R_t , is defined as the cumulative future reward from that point until the end of the decision horizon T : $R_t = \sum_{i=t}^T r_i$. Conditioning on return-to-go rather than immediate rewards enables the model to associate state-action sequences with desirable long-term outcomes [6]. We extend this framework to the multi-agent setting, where the state, action, and reward of the i^{th} agent at each timestep are denoted as s_i, a_i, r_i , and its trajectories $\tau : (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T, a_T, r_T)$ consist of sequences of state, action, and reward tuples. To align with return-conditioned learning, these trajectories are restructured as $\tau = (R_0, s_0, a_0, R_1, s_1, a_1, \dots, R_T, s_T, a_T)$, where R_i is the return-to-go from timestep i onward.

3.4 S4-based Agent

This section provides a brief background on State Space Models (SSMs) and Structured State Space Models (S4), which form the basis for designing individual agents in our framework.

SSMs are a classical framework for modeling sequential data through latent dynamical systems. At each timestep t , the model receives an input $u(t)$, updates an internal memory or hidden state $x(t)$, and generates an output $y(t)$ using a first-order linear time-invariant differential equation:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \quad (2)$$

SSMs typically operate on continuous time sequences where A, B, C, D are parameter matrices of appropriate dimensions. To apply SSMs in discrete-time domains, such as typical reinforcement learning settings, the continuous-time formulation is discretized with a fixed step size Δ using schemes like the bilinear transform [36], yielding the discretized linear recurrence:

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

where the discretized matrices $\bar{A}, \bar{B}, \bar{C}, \bar{D}$ are functions of the continuous-time parameters and the step size Δ .

S4 models [11] extend this formulation with efficient parameterizations to enable stable and scalable modeling of long sequences. S4 leverages special initialization strategies such as HiPPO [10] to preserve long-range dependencies, and its structured formulation allows it to be integrated effectively with deep learning models for sequence modeling tasks. Various techniques have since been developed to improve the model’s performance, stability, and training efficiency [12, 13]. Similar to these works, D is represented here by a skip connection.

Importantly, since Eq. 3 is linear and time-invariant, the output sequence y can be computed directly in parallel based on the input sequence u via non-circular convolution with a kernel \bar{K} :

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

where \bar{K} is a function of $\bar{A}, \bar{B}, \bar{C}, \bar{D}$ and context length L which is pre-fixed during training. This convolution can be computed efficiently across all time steps, allowing for parallelizable training. The recurrent view of the SSM also allows for faster inference with low memory. This is a key advantage over transformers, which makes the use of SSMs very effective in reinforcement learning settings, which require faster inference for the collection of online interactions with the environment. Additional details can be found in the Appendix.

In this work, each agent is modeled using an S4-based architecture that processes sequences of past information and target returns-to-go to predict the next action. We then extend this design to the multi-agent setting by enabling cooperation through information sharing, as detailed below.

3.5 Information Sharing with Limited Communication

To facilitate scalable cooperation among S4-based agents, we introduce a communication mechanism restricted to consecutive agents within the SE-MDP sequence. Specifically, each agent’s memory information—captured by the hidden state of its S4 module—is passed to the next agent in the sequence. As a result, the hidden state of an agent in the sequence implicitly encodes information from all preceding agents. This design induces a bidirectional flow of information and dependency across agents. At each timestep t , a projection of the previous agent’s latent state, h_{i-1}^t , is provided as an additional input to the next agent, alongside its own input \hat{u}_i^t ,

thereby influencing both its action a_i^t and updated memory state h_i^t :

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

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

$$\frac{\partial J}{\partial \theta_i} = \frac{\partial J}{\partial a_i^t} \cdot \frac{\partial a_i^t}{\partial \theta_i} + \frac{\partial J}{\partial a_{i+1}^t} \cdot \frac{\partial a_{i+1}^t}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial \theta_i} \quad (6)$$

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 due to the quadratic growth in memory requirements during training and execution, our mechanism is highly efficient, requiring only constant memory per agent.

We first adapt Decision S4 (DS4) for each agent with parameter sharing, similar to the Multi-Agent Decision Transformer (MADT) [26]. Unlike MADT, however, the Multi-Agent Decision S4 (MADS4) is trained in a sequentially dependent manner, where agents can share accumulated memory information with the next agent in the sequence. The offline version of MADS4, trained on pre-collected trajectories, is detailed in the next section. These pre-trained models can be further fine-tuned in an on-policy setting using MAPPO [49].

4 MULTI-AGENT DECISION S4 (MADS4)

4.1 MADS4: Offline Training

Input formulation. In offline MADS4 training, each agent learns from trajectories of past observations, its past actions, the preceding agent’s latent state, and returns-to-go. As in MADT, the state of each agent at each time step s_i^t may include global state s_{gi}^t and local observation o_i^t , though we observe similar performance without global state inputs, as shown in the Appendix. Thus, the S4 input trajectory for agent i consists of:

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

where R^t is the returns-to-go from current time step t , s_{gi}^t is the current shared global state, o_i^t is the current local observation, a_i^{t-1} is the previously executed action of the i^{th} agent, h_{i-1}^t is the current hidden state representation of the preceding agent. The model is trained to predict actions (action logits) at time step t in an autoregressive manner based on the data seen so far. The output of the S4-based model is used as the action probabilities which are sampled after applying the action availability masks.

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

where θ are the parameters of the MADS4. In this work, parameter sharing is allowed across the agents for training stability, and

thus, essentially, a single model is trained, which takes into account different inputs for different agents along with their specific one-hot agent IDs.

Algorithm 1 MADS4-Offline Training

Input: Offline dataset $\mathcal{D} : \{\tau_i : \langle s_{gi}^t, o_i^t, a_i^t, v_i^t, d_i^t, R_i^t \rangle_{t=1}^n\}_{i=1}^n$, where n is the number of agents, and v_i^t denotes the available actions for the i^{th} agent at time t ; d_i^t denotes the done signal for an episode

Initialize α as the learning rate, L as the context length

Initialize θ for the S4 models based on HIPPO initialization

- 1: **for** $i = 1 : n$ **do** ▷ Iterate over each agent
- 2: From τ_i and h_{i-1}^t , create $X_i : \{R_i^t, s_{gi}^t, o_i^t, a_i^{t-1}, h_{i-1}^t\}_{t=1}^T$, where h_{i-1}^t is the latent state representation of the previous agent, assuming $h_{i-1}^0 = \mathbf{0}$
- 3: Zero-pad the trajectory to a constant length L when d_i^t is true ▷ Pad when the agent is done
- 4: Compute action output sequence $\hat{a}_i = \{\hat{a}_i^1, \dots, \hat{a}_i^T\}$ and latent state projections $h_i = \{h_i^1, \dots, h_i^T\}$
- 5: **for** $t = 1 : T$ **do** ▷ Loss calculation over time steps
- 6: Mask illegal actions via $P(\hat{a}_{ij}^t | \tau_i^{\leq t}; \theta) = 0$ if v_{ij}^t is False, where j is the unavailable action index
- 7: Predict the action $\hat{a}_i^t = \arg \max_j P(\hat{a}_{ij}^t | \tau_i^{\leq t}; \theta)$
- 8: **end for**
- 9: Update θ :

$$\theta \leftarrow \arg \max_{\theta} \frac{1}{T} \sum_{t=1}^T P(a_i^t) \log P(\hat{a}_i^t | \tau_i^{\leq t}; \theta)$$

10: **end for**

Return: θ

Network architecture and training. The MADS4 architecture consists of three key components, as illustrated in Figure 1(b): (i) Input, state, and output projection layers, each consisting of a fully connected layer followed by ReLU activations; (ii) Input encoder layers, which handle states, actions, and rewards/returns, each implemented as fully connected linear layers; (iii) Sequence modeling component which consists of stacked S4 blocks, where each block consists of Batch Normalization layer followed by S4 layer, linear mixing layer with GELU activation, and a dropout layer. We employed the HIPPO-initialized "Normal Plus Low Rank" S4 kernel with input channel size $H = 96$ and the S4 state size $N = 96$. The effect of varying state and input sizes is shown in Section 5.4.

In the offline setting, the S4 model is trained efficiently using the convolutional view on randomly sampled full trajectories from the offline dataset, zero-padded to a fixed context length. Unlike Transformers, whose self-attention incurs quadratic time and memory costs that constrain context length, S4 scales more favorably and can therefore be trained on full trajectories that are often much longer than those used in Transformer-based offline RL. The impact of truncating trajectory lengths on the model performance is shown in Section 5.4. Actions are predicted from the model's action logits, and training minimizes cross-entropy between predicted and ground-truth actions.

Algorithm 2 MADS4: On-policy Fine-tuning

- 1: Copy the model weights θ to the actor or policy network $\pi : \pi(u_i)$, where $u_i = \{R_i^t, s_{gi}^t, o_i^t, a_i^{t-1}, h_{i-1}^t\}$; Initialize ϕ , the parameters for critic V
- 2: Set learning rates α_{π}, α_V for actor and critic
- 3: **for** iterations = 1, M **do**
- 4: Set data buffer $D = \{\}$
- 5: **for** $i = 1$ to $batch_size$ **do**
- 6: $\tau = []$ ▷ Empty list
- 7: Initialize $h_0^{(1)}, \dots, h_0^{(n)}$ actor S4 states
- 8: **for** each timestep t in the environment **do**
- 9: **for** agent = 1 : n **do**
- 10: $p_i^t, h_i^t = \pi(u_i^t, h_{i-1}^t; \theta)$
- 11: Sample $a_i^t \sim p_i^t$
- 12: **end for**
- 13: Execute actions \mathbf{a}^t , observe $r^t, s_{g_i}^{t+1}, \mathbf{o}^{t+1}$
- 14: $\tau += [s_t, \mathbf{o}_t, \mathbf{h}_t, \mathbf{a}_t, r_t, R_t, s_{t+1}, \mathbf{o}_{t+1}]$
- 15: **end for**
- 16: Calculate advantage A^t via GAE on τ and store τ with A^t in the buffer D
- 17: **end for**
- 18: **for** $k = 1, \dots, K$ training steps **do**
- 19: Sample batch from replay-buffer $B = \{(s_{g_i}^t, \mathbf{o}_t, a^{t-1}, R^{t-1}, h^{t-1}, s_{g_i}^{t+1}, \mathbf{o}^{t+1}, a^t, h^t, r^t, R^t, A^t)\} \subset D$ for each agent
- 20: Calculate Bellman target estimate: $y = r^t + \gamma V(s^{t+1}, h^t)$
- 21: Update critic: $\phi_V = \phi_V - \alpha_V \nabla_{\phi_V} (V(s^t, h^{t-1}) - y)^2$
- 22: If actor freezing is over, update actor:
- 23: $\theta_{\pi} \leftarrow \arg \max_{\theta_{\pi}} \mathbb{E}_{s \sim \rho_{\theta_{\text{old}}}, a \sim \pi_{\theta_{\text{old}}}} [\text{clip}(w, 1 - \epsilon, 1 + \epsilon) A^t]$
- 24: **end for**

$$\text{where the importance weight } w = \frac{\pi_{\theta}(a_i | o_i)}{\pi_{\theta_{\text{old}}}(a_i | o_i)}$$

4.2 MADS4: On-policy Fine-tuning

For online fine-tuning, we initialize with the offline-pretrained agent and continue training on-policy. Agents collect rollouts into a buffer containing local observations/actions, shared global states, rewards, returns-to-go, and also the S4 latent states. Under the MAPPO actor-critic framework [49], the offline-pretrained S4 model serves as the actor which predict actions via the recurrent view based on latent states and other inputs as used in the pretraining stage. The critic is conditioned on the global states of the environment and the encoded latent S4 states to evaluate the value function.

Network architecture and training. The pre-trained model is loaded as the actor network, which predicts action probabilities and next states as:

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

The critic network is parameterized by fully connected layers with ReLU activations, which take the shared global state 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. This heuristic improves exploration efficiency in practice, and may be further optimized for different environments. Additional details on the experimental setup and training are provided in the Appendix.

5 EXPERIMENTS

5.1 Datasets and Baselines

We evaluate the performance of MADS4 on challenging cooperative MARL benchmarks of Multi-Robot Warehouse (RWARE) [29] and StarCraft2 Multi-Agent Challenge (SMAC) [31]. The offline datasets in the RWARE domain are obtained from [25], which consists of diverse trajectories collected by training Multi-Agent Transformer (MAT) [40]. The RWARE datasets consist of expert policies trained on 2 maps (tiny and small) with different numbers of agents. For the SMAC domain, the datasets provided by [26] have been used, which consists of trajectories collected with online trained MAPPO agents. The datasets consist of three trained quality levels of the agents, good, medium, and poor, tested on the different SMAC maps. For this work, we chose four representative maps consisting of two hard (5m vs. 6m, 2c vs. 64zg) and two super hard (6h vs. 8z, corridor) maps for evaluating MADS4. Additional statistics on the offline datasets can be found in the Appendix.

For comparisons on both domains, we compare with several recent offline MARL algorithms from the paradigms of both offline reinforcement learning and sequence-based supervised learning. The offline RL baselines considered for comparison are Behaviour Cloning (BC) [9], OptiDICE [18], AlberDICE [25], ICQ [47], OMAR[28] and OMIGA[39]. The sequence-based learning algorithms considered in this work include MADT [26] and MADTKD [35], which are based on transformers. MADT policies do not involve any cooperation during learning, whereas MADTKD incorporates a degree of cooperation distilled into the agents from the centralized teacher model.

5.2 Offline Training

Here, we compare the performance of offline trained MADS4, where the agents share information in the form of their latent state projections. The trained agents are deployed on the online RWARE and StarCraft2 environments for evaluation. Tables 1 and 2 show the mean and standard deviation of average returns in RWARE and SMAC domains, respectively, evaluated over 30 episodes and 5 different training seeds. During evaluation, the desired returns-to-go is set at 10% higher than the highest returns encountered in the offline datasets.

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. In this domain, MADS4 outperforms all baselines across the evaluated maps, achieving statistically significant improvements, with a larger performance gap observed on the small and tiny 6-agent maps, where tight coordination is essential to prevent 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 demonstrates consistent performance that is similar to or better than the considered baselines across all studied maps. Notably, the model outperforms all baselines in the hard and superhard maps, specifically in the 2c vs. 64zg and 6h vs. 8z scenarios.

5.3 On-policy Fine-tuning

We evaluate whether the performance of offline pre-trained models can be enhanced through on-policy fine-tuning. During this phase, MADS4 interacts with the environment, collecting trajectories that are stored in a buffer and used to update the S4-based models via recurrence. As shown in Figure 2, on-policy training builds upon and improves the offline pretraining results. Furthermore, on-policy training without pretraining consistently results in sub-optimal performance across all tasks, underscoring the importance of pretraining for achieving superior results.

However, prolonged on-policy training can sometimes degrade the performance of pre-trained models, as shown in the Appendix. This degradation likely arises from the inherent instability of training S4 modules in a recurrent setup, compared to the more stable convolution-based operations employed during offline pretraining, leading to error accumulation. We mitigate this issue by freezing the S4 kernel parameter A , which governs state-to-state transitions independent of inputs, and fine-tuning only the input-dependent parameters B and C .

5.4 Ablation Study and Hyperparameter Analysis

Effect of sharing information. The sharing of information between agents leads to significantly improved cooperative behavior, as reflected in the higher average rewards shown in Figure 3. This performance boost is particularly pronounced in more complex tasks that involve a greater number of agents and demand precise coordination. Importantly, this method of sharing information is scalable, where an agent only needs to communicate with the next agent, minimizing overhead while ensuring efficient coordination. Information can be shared in multiple forms, namely by passing the action logit outputs from the S4 model, the latent state representations, or a combination of both from the preceding agent. A more detailed analysis of how different types of shared information impact performance is provided in the Appendix.

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. Figure 4(a) shows similar

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.

SMAC Map	Data	RL-based			Sequence-based	
		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

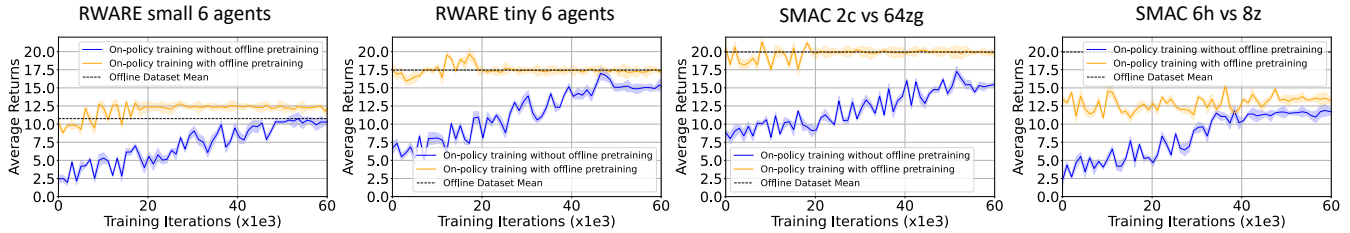


Figure 2: Training curves of on-policy training of MADS4 with and without offline pretraining. Mean and standard deviations of average returns are plotted over 5 independent runs.

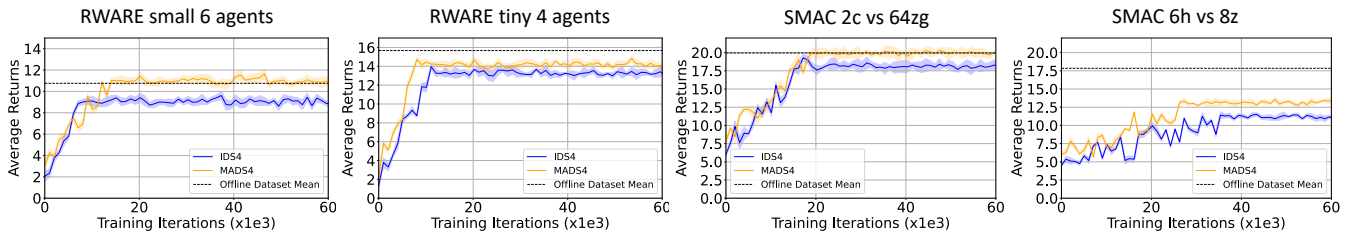


Figure 3: Training curves of MADS4 with information sharing between consecutive agents and IDS4 where agents are trained independently. Mean and standard deviations of average returns are plotted over 5 independent runs.

performance, demonstrating that MADS4 is robust to agent ordering within the SE-MDP framework. Notably, our approach only

requires each agent to communicate with one unique peer which can be selected randomly to ensure that every agent’s information

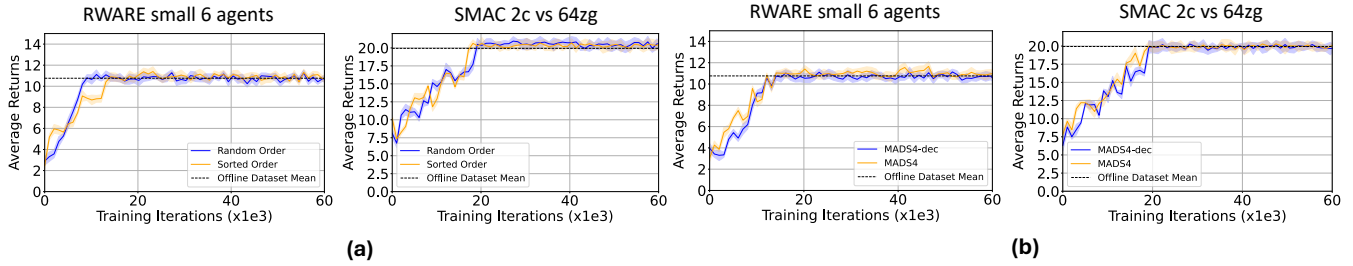


Figure 4: (a) The effect of having a shuffled random order vs. a fixed sorted order of the agents in the SE-MDP framework on the RWARE domain in the small 6 agents scenario and SMAC domain in the 2c vs. 64zg map. (b) The comparison of the performance of MADS4 vs. MADS4-dec (decentralized MADS4) on RWARE small map with 6 agents and SMAC map 2c vs 64 zg.

Table 3: Average returns for different MADS4 model configurations on the RWARE Small map. Each of the models is denoted by (i) N , the S_4 state size, and (ii) H , the number of input/output channels. For parameter efficiency, we report the percentage of parameters used relative to MADTKD.

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

is passed across the network without the need for any centralized optimization or sophisticated coordination.

MADS4 in decentralized setting. To adapt MADS4 for a decentralized setting, where agents act in parallel, we use each agent’s hidden state from the previous timestep as a proxy for the current timestep. Since all actions at $t - 1$ are finalized before acting at t , the memory information of all agents is readily available for use. 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 4(b). This modification enables our algorithm to function effectively in decentralized policy settings without performance degradation.

S_4 model size parameters. We analyze the impact of the S_4 model size parameters, specifically the number of input channels (H) and the latent state size (N), on the model performance, as shown in Table 3. We compare the total number of parameters against the 1.8 million parameters reported for MADTKD in [35]. Our biggest model with $N=96$ and $H=96$ was used in all our experiments, which consists of about 200k parameters.

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 5, 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.

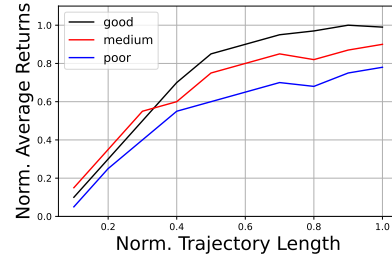


Figure 5: The effect of truncating the trajectory length during training. The average returns are normalized with the maximum returns encountered in the offline dataset.

6 CONCLUSIONS

In this work, we demonstrate the effectiveness of S_4 -based models in outperforming transformer-based architectures for sequence-to-sequence offline multi-agent reinforcement learning (MARL) tasks. By structuring agent interactions within the SE-MDP framework and limiting communication to the exchange of information between unique, arbitrarily chosen agent pairs, MADS4 enables more scalable and efficient cooperation. This contrasts with state-of-the-art offline RL and centralized transformer-based models, which require complete access to all agents’ information during training, leading to scalability challenges. Additionally, MADS4 offers a low-latency and lightweight model that can be trained more efficiently than transformers and fine-tuned online using recurrent computations.

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation through the Scipe AI in Civil Engineering Fellowship (Award No. 2321040).

REFERENCES

- [1] Chenjia Bai, Lingxiao Wang, Zhuoran Yang, Zhihong Deng, Animesh Garg, Peng Liu, and Zhaoran Wang. 2022. Pessimistic bootstrapping for uncertainty-driven offline reinforcement learning. *arXiv preprint arXiv:2202.11566* (2022).
- [2] Shmuel Bar-David, Itamar Zimmerman, Eliya Nachmani, and Lior Wolf. 2023. Decision s4: Efficient sequence-based rl via state spaces layers. *arXiv preprint arXiv:2306.05167* (2023).
- [3] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).
- [4] Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. 2022. Recurrent memory transformer. *Advances in Neural Information Processing Systems* 35 (2022), 11079–11091.
- [5] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. 2012. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial Informatics* 9, 1 (2012), 427–438.
- [6] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems* 34 (2021), 15084–15097.
- [7] Peng Cheng, Xianyuan Zhan, Wenjia Zhang, Youfang Lin, Han Wang, Li Jiang, et al. 2024. Look beneath the surface: Exploiting fundamental symmetry for sample-efficient offline rl. *Advances in Neural Information Processing Systems* 36 (2024).
- [8] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2017. Counterfactual multi-agent policy gradients (2017). *arXiv preprint arXiv:1705.08926* (2017).
- [9] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*. PMLR, 2052–2062.
- [10] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. 2020. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems* 33 (2020), 1474–1487.
- [11] Albert Gu, Tri Dao Goel, Avanika Rudra, and Christopher Ré. 2022. Efficiently Modeling Long Sequences with Structured State Spaces. In *International Conference on Learning Representations (ICLR)*. <https://openreview.net/forum?id=uYLFoz1vIAC>
- [12] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. 2021. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems* 34 (2021), 572–585.
- [13] Ankit Gupta, Albert Gu, and Jonathan Berant. 2022. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems* 35 (2022), 22982–22994.
- [14] Jiechuan Jiang and Zongqing Lu. 2023. Offline Decentralized Multi-Agent Reinforcement Learning. In *ECAI*. 1148–1155.
- [15] Ilya Kostrikov, Rob Fergus, Jonathan Tompson, and Ofir Nachum. 2021. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*. PMLR, 5774–5783.
- [16] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. 2019. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in neural information processing systems* 32 (2019).
- [17] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 1179–1191.
- [18] Jongmin Lee, Wonseok Jeon, Byungjun Lee, Joelle Pineau, and Kee-Eung Kim. 2021. Optidice: Offline policy optimization via stationary distribution correction estimation. In *International Conference on Machine Learning*. PMLR, 6120–6130.
- [19] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643* (2020).
- [20] Chuming Li, Jie Liu, Yinmin Zhang, Yuhong Wei, Yazhe Niu, Yaodong Yang, Yu Liu, and Wanli Ouyang. 2023. Ace: Cooperative multi-agent q-learning with bidirectional action-dependency. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 37. 8536–8544.
- [21] Jianxiong Li, Xianyuan Zhan, Haoran Xu, Xiangyu Zhu, Jingjing Liu, and Ya-Qin Zhang. 2022. When data geometry meets deep function: Generalizing offline reinforcement learning. *arXiv preprint arXiv:2205.11027* (2022).
- [22] Michael L. Littman. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*. Elsevier, 157–163.
- [23] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems* 30 (2017).
- [24] Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Foerster, Sander Singh, and Feryal Behbahani. 2024. Structured state space models for in-context reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2024).
- [25] Daiki E Matsunaga, Jongmin Lee, Jaeseok Yoon, Stefanos Leonardos, Pieter Abbeel, and Kee-Eung Kim. 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 (2023), 72648–72678.
- [26] Linghui Meng, Muning Wen, Yaodong Yang, Chenyang Le, Xiyun Li, Weinan Zhang, Ying Wen, Haifeng Zhang, Jun Wang, and Bo Xu. 2021. Offline pre-trained multi-agent decision transformer: One big sequence model tackles all smac tasks. *arXiv preprint arXiv:2112.02845* (2021).
- [27] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. 2008. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research* 32 (2008), 289–353.
- [28] Ling Pan, Longbo Huang, Tengyu Ma, and Huazhe Xu. 2022. Plan better amid conservatism: Offline multi-agent reinforcement learning with actor rectification. In *International conference on machine learning*. PMLR, 17221–17237.
- [29] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. 2020. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. *arXiv preprint arXiv:2006.07869* (2020).
- [30] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2020. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research* 21, 178 (2020), 1–51.
- [31] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. 2019. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043* (2019).
- [32] Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. 2022. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933* (2022).
- [33] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. 2019. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*. PMLR, 5887–5896.
- [34] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. 2017. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296* (2017).
- [35] Wei-Cheng Tseng, Tsun-Hsuan Johnson Wang, Yen-Chen Lin, and Phillip Isola. 2022. Offline multi-agent reinforcement learning with knowledge distillation. *Advances in Neural Information Processing Systems* 35 (2022), 226–237.
- [36] Arnold Turing. 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 (1947), 130–142.
- [37] Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, et al. 2023. Jumpstart reinforcement learning. In *International Conference on Machine Learning*. PMLR, 34556–34583.
- [38] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. 2020. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062* (2020).
- [39] Xiangsen Wang, Haoran Xu, Yinan Zheng, and Xianyuan Zhan. 2024. Offline multi-agent reinforcement learning with implicit global-to-local value regularization. *Advances in Neural Information Processing Systems* 36 (2024).
- [40] Muning Wen, Jakob Kuba, Runji Lin, Weinan Zhang, Ying Wen, Jun Wang, and Yaodong Yang. 2022. Multi-agent reinforcement learning is a sequence modeling problem. *Advances in Neural Information Processing Systems* 35 (2022), 16509–16521.
- [41] Yifan Wu, George Tucker, and Ofir Nachum. 2019. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361* (2019).
- [42] Yue Wu, Shuangfei Zhai, Nitish Srivastava, Joshua Susskind, Jian Zhang, Ruslan Salakhutdinov, and Hanlin Goh. 2021. Uncertainty weighted actor-critic for offline reinforcement learning. *arXiv preprint arXiv:2105.08140* (2021).
- [43] Haoran Xu, Li Jiang, Li Jianxiong, and Xianyuan Zhan. 2022. A policy-guided imitation approach for offline reinforcement learning. *Advances in Neural Information Processing Systems* 35 (2022), 4085–4098.
- [44] Haoran Xu, Li Jiang, Jianxiong Li, Zhuoran Yang, Zhaoran Wang, Victor Wai Kin Chan, and Xianyuan Zhan. 2023. Offline rl with no ood actions: In-sample learning via implicit value regularization. *arXiv preprint arXiv:2303.15810* (2023).
- [45] Haoran Xu, Xianyuan Zhan, Jianxiong Li, and Honglei Yin. 2021. Offline reinforcement learning with soft behavior regularization. *arXiv preprint arXiv:2110.07395* (2021).

- [46] Haoran Xu, Xianyuan Zhan, and Xiangyu Zhu. 2022. Constraints penalized q-learning for safe offline reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 8753–8760.
- [47] Yiqin Yang, Xiaoteng Ma, Chenghao Li, Zewu Zheng, Qiyuan Zhang, Gao Huang, Jun Yang, and Qianchuan Zhao. 2021. Believe what you see: Implicit constraint approach for offline multi-agent reinforcement learning. *Advances in Neural Information Processing Systems* 34 (2021), 10299–10312.
- [48] Dayon Ye, Minji Zhang, and Yu Yang. 2015. A multi-agent framework for packet routing in wireless sensor networks. *sensors* 15, 5 (2015), 10026–10047.
- [49] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. 2021. The surprising effectiveness of PPO in cooperative, multi-agent games (2021). *arXiv preprint arXiv:2103.01955* (2021).
- [50] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. 2020. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems* 33 (2020), 14129–14142.