Scalable Multi-Agent Model-Based Reinforcement Learning

Vladimir Egorov
JetBrains Research; HSE University
Saint Petersburg, Russian Federation
vsegorov@edu.hse.ru

Alexei Shpilman
JetBrains Research; HSE University
Saint Petersburg, Russian Federation
alexey@shpilman.com

ABSTRACT
Recent Multi-Agent Reinforcement Learning (MARL) literature has been largely focused on Centralized Training with Decentralized Execution (CTDE) paradigm. CTDE has been a dominant approach for both cooperative and mixed environments due to its capability to efficiently train decentralized policies. While in mixed environments full autonomy of the agents can be a desirable outcome, cooperative environments allow agents to share information to facilitate coordination. Approaches that leverage this technique are usually referred as communication methods, as full autonomy of agents is compromised for better performance. Although communication approaches have shown impressive results, they do not fully leverage this additional information during training phase. In this paper, we propose a new method called MAMBA which utilizes Model-Based Reinforcement Learning (MBRL) to further leverage centralized training in cooperative environments. We argue that communication between agents is enough to sustain a world model for each agent during execution phase while imaginary rollouts can be used for training, removing the necessity to interact with the environment. These properties yield sample efficient algorithm that can scale gracefully with the number of agents. We empirically confirm that MAMBA achieves good performance while reducing the number of interactions with the environment up to an orders of magnitude compared to Model-Free state-of-the-art approaches in challenging domains of SMAC and Flatland.

KEYWORDS
Multi-Agent Reinforcement Learning; Model-Based Reinforcement Learning; Communication

ACM Reference Format:
Vladimir Egorov and Alexei Shpilman. 2022. Scalable Multi-Agent Model-Based Reinforcement Learning. In Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022, IFAAMAS, 10 pages.

1 INTRODUCTION
Reinforcement Learning (RL) has achieved numerous breakthroughs [7, 62, 72] in recent years both for single- and multi-agent environments. However, sample efficiency has always been a prominent issue as a tremendous number of interactions is required by Model-Free Reinforcement Learning (MFRL) approaches to achieve optimal performance. Model-Based Reinforcement Learning (MBRL) tries to alleviate this issue by allowing the agent to interact with a learned model of the environment instead. It has shown success both in environments with known world dynamics such as shogi or go [57] and where dynamics need to be learned from samples such as Atari [25, 33]. However, translating this success to multi-agent settings is non-trivial. Several studies [39, 60] have shown that MBRL approaches can be much more sample efficient than their MFRL counterparts in environments with two agents. Nonetheless, all of the current state-of-the-art MFRL approaches on popular multi-agent benchmarks such as StarCraft Multi-Agent Challenge [56] remain uncontested.

One of the major paradigms in Multi-Agent Reinforcement Learning is Centralized Training with Decentralized Execution (CTDE) [18, 47, 54], which holds to the premise that we can use additional information during training as long as execution remains decentralized. This paradigm is well-motivated when we want full autonomy for the agents [48], for example, when the task is not fully cooperative because agents might not want to share their information with others [8, 9, 51]. Furthermore, in many real world scenarios, such as autonomous driving [14] or drone management [3], agents must collaborate with each other to achieve their goals and they can also share information for better coordination. This setup is usually defined as a communication technique [20, 46], as the agents no longer have full autonomy yet they can leverage the messages from others to make their decisions. As a part of CTDE, communication methods usually restrict the message channel either by limiting bandwidth or receiving messages only from the neighbours in the environment [21]. Other CTDE techniques can still be applied together with the communication technique to stabilize learning or solve the credit assignment issue. Note that if we decide to centralize execution even more, reducing the multi-agent problem to single-agent, this solution will suffer from scalability issues as the number of states and actions will grow exponentially with the number of agents. Thus, communication methods strike a balance between autonomy of the agents and efficiently using available information to make decisions in collaborative environments. However, previous studies do not fully leverage this additional shared information during the training phase.

In this paper, we show that communication can be sufficient to sustain a world model, making it possible to exploit centralized training further by training policies without interacting with the environment. Furthermore, it is sufficient to only communicate discrete messages with the neighbouring agents to update a world model during execution phase. We denote our approach as MAMBA, which is derived from Multi-Agent Model-Based Approach. To our knowledge, this is the first method that extends world models to environments with an arbitrary number of agents and that views world models as an instance of communication. We show that our method can reduce the required number of samples by an order of magnitude compared to MFRL approaches and consistently

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascari, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online. © 2022 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamais.org). All rights reserved.
outperform current state-of-the-art methods in low data regime in challenging environments of SMAC [56] and Flatland [50].

2 DEFINITIONS AND BACKGROUND

2.1 Markov Game

A Markov game [44] is a tuple \( G = (S, n, A_i, O, T, r) \), where \( S \) is a set of states, \( n \) is a number of players, \( A_i \) is a set of actions of player \( i \), \( O : S \times O \rightarrow \mathbb{R}^d \) is an observation function that specifies \( d \)-dimensional observations available to each agent and \( O_i \) is a set of observations \( o \) of agent \( i \), \( T : S \times A_1 \times \cdots \times A_n \rightarrow \Delta(S) \) and \( r_i : S \times A_1 \times \cdots \times A_n \rightarrow \mathbb{R} \) are transition and reward functions respectively, where \( \Delta \) is a discrete probability distribution over \( S \). We define \( \Delta_0(S) \) as the distribution of initial states and return of player \( i \) in state \( s \) with discount factor \( \gamma \in [0, 1) \) as \( \pi_i(s) = \sum_{t=0}^{\infty} \gamma^t r_i(s_t, a_t^1, \ldots, a_t^n) \mid s_0 = s \). Each player has its own policy \( \pi_i : O_i \rightarrow \Delta(\mathcal{A}_i) \) that returns probability for each action to be taken when observing \( o_i \) and value function \( V_i(s) = \mathbb{E}_{\pi_i}[R_i(s)] \). We also define Q-function as \( Q_i(s,a) = \mathbb{E}_{\pi_i}[R_i(s) \mid a_i = a] \). Bold font will denote vector, e.g. \( a = (a_1, \ldots, a_n) \) and subscript \( a \) will denote current step in the environment.

2.2 Single-Agent Reinforcement Learning

Model-Free Reinforcement Learning. There are two major MFRL frameworks, namely Q-learning [75] and Actor-Critic [49]. In Actor-Critic framework, the actor’s goal is to learn a policy \( \pi(s) \) that maximizes agents returns predicted by the Critic. A widely-used method is proximal policy optimization (PPO) [59], where the Actor’s neural network is trained on the following loss:

\[
\mathcal{L}_\pi(A_t) = - \min(\mathcal{R}_t A_t, \text{clip}(\mathcal{R}_t, 1 - \epsilon, 1 + \epsilon) A_t) \tag{1}
\]

where \( \mathcal{R}_t = \frac{\pi(s_t)}{\pi_0(s_t)} \) denotes probability ratio of the policies after and before the update and \( A_t \) is an advantage function predicted by the Critic. Using this loss ensures that the agent’s policy makes a conservative update within a trust region. The advantage is defined as \( A_t = y_t - V_t \), where \( y_t = r_t + \gamma V_{t+1} \). The Critic’s neural network is independently trained to predict \( V_t \) by minimizing squared TD error \( (y_t - V_t)^2 \). In Q-learning framework, we do not have a policy \( \pi(s) \) and instead use learned Q-function directly to predict next action.

Model-Based Reinforcement Learning. MBRL approaches [68] additionally learn a world model that includes both transition function \( T \) and reward function \( r \). Decoupling learning dynamics of the system from the decision making allows us to use more sophisticated unsupervised learning techniques to represent current state in latent space as well as to use imaginary rollouts for training [25, 33]. Many previous works provide both theoretical and empirical evidence that MBRL approaches can be much more sample efficient than their MFRL counterparts. Recent breakthroughs of Dreamer [23, 25] also showed that MBRL is capable of achieving same or better performance than Model-Free approaches both in continuous and discrete environments.

2.3 Multi-Agent Reinforcement Learning

In Multi-Agent Reinforcement Learning (MARL), multiple agents learn and interact in the same environment. In this paper, we will focus on cooperative environments [56], where agents have the same goal and they need to collaborate with each other to achieve it. One naive approach in MARL is training agents independently using unmodified single-agent RL techniques [13, 70]. However, it does not have any convergence guarantees due to the inherent non-stationarity of multi-agent environments resulting from the constantly changing policies of other agents [27, 41]. On the other side of the spectrum, we could look at the multi-agent problem as a single-agent one. This yields a fully centralized approach that controls all agents simultaneously based on global information. Consequently, a centralized solution cannot scale to a large number of agents due to the resultant exponential growth of the action space, known as the familiar ‘curse of dimensionality’ [67]. This leaves us with one viable technique to train agents in multi-agent environments, namely Centralized Training with Decentralized Execution. We also explicitly highlight the communication technique, which is considered part of CTDE [21], that allows an additional message channel during execution for better coordination.

Centralized Training with Decentralized Execution. CTDE is a combination of independent and centralized MARL [18, 38, 47, 54, 64, 67]. This paradigm allows additional information to be used in training provided that execution is still decentralized. This enhancement can help to alleviate the issues of credit assignment and non-stationarity as we no longer need to rely only on local observations and rewards during training. Decentralized execution also ensures that we deal with the curse of dimensionality as agents still make their decisions independently based on relevant local information.

Communication can be seen as a part of the CTDE paradigm as we can still use additional information on the training stage, but now we allow agents to communicate with each other during execution, which is a natural assumption for cooperative environments [20, 46, 66] where agents have the same goal and can also be used in mixed environments [63] to promote cooperation. Despite being considered CTDE [21], we argue that communication can greatly benefit from possessing certain properties to ensure decentralized execution. In this paper, we will focus on the two desirable properties of communication that can greatly improve scalability and decentralization of the agents. The first property is called discrete communication and is strongly connected with the limitation of the message channel bandwidth as it restricts agents to sending each other only discrete messages [20, 21]. This property allows us to significantly reduce the number of bits for message transmission and can be considered a universal restriction between the environments on the number of bits allowed for one message. The second property is called locality [80], which can be naturally defined in most environments. This property allows an agent to communicate only with its neighbours on the map, which makes the decision making decentralized since each agent takes action individually based only on local information [80]. Unfortunately, some environments do not exhibit this locality property either because the map is too small or coordination between all agents is required to solve the task.
Multi-Agent Deep Deterministic Policy Gradient (MADDPG). [47] MADDPG is a CTDE algorithm specifically designed for mixed environments. MADDPG shows that augmenting critics with the states and actions of all agents can reduce the variance of policy gradients, which can improve performance.

Counterfactual Multi-Agent policy gradients (COMA). [18] COMA is an Actor-Critic algorithm designed to alleviate the credit assignment issue in multi-agent environments. The core idea is based on Difference Rewards [77], which compares the current global reward of the agent to the average reward that the agent can get in the current state. COMA applies this idea to Q-functions of agents, additionally using augmented critics similar to MADDPG.

QMIX. [54] QMIX is an algorithm designed for training Q-learning agents in cooperative environments. As with COMA, it tries to solve credit assignment using additional information during training. As agents still need the Q-function during execution, we cannot alter its input as in COMA. QMIX proposes to use a mixer function to combine Q-functions of agents into a centralized Q-function that is trained on global reward. QMIX ascertains that mixer function ensures monotonicity in its inputs, which makes the argmax of the agents’ Q-functions consistent with the argmax of the centralized Q-function. This property is called Individual-Global-Max (IGM) and is essential for factorizing the global Q-function to agents’ Q-Functions.

QTRAN. [64] QTRAN is an algorithm that extends the class of functions that we can factorize compared to QMIX. QTRAN shows that the monotonicity property can be lifted from the mixer function and introduces additional “track” losses that keep the agents’ Q-functions close to the global Q-function, which is sufficient to preserve the IGM property but introduces additional training complexity.

CommNet. [66] CommNet is a communication algorithm that employs continuous protocols. CommNet uses LSTM [29] to process messages and shows that communication can be essential in solving tasks in partially observable environments.

G2A. [46] G2A is an algorithm that uses two-stage attention architecture to process messages from other agents. The first stage uses Hard Attention to detect relevant agents in the environment, while the second stage uses Soft Attention to process messages from these agents.

2.4 Attention Mechanism

Attention [5, 71] has become an ubiquitous approach for sequence data processing, achieving state-of-the-art performance in multitude of fields such as NLP [15] and CV [16]. It has also shown to be useful in MARL when processing input observations of all agents [72] or solving credit assignment [30].

In this paper, we will focus mainly on self-attention, which takes as an input value matrix \( V \) with dimensionality \( n \times d \) and for each of \( n \) vectors outputs weighted average of the whole vector sequence. There are two types of attention mechanism, namely soft attention and hard attention [61]. Following [71], the former assigns the weights for each value vector using this formula:

\[
\text{Soft Self-Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d}} \right) V
\]

where \( Q, K, V \) are linear projections of the value vectors, and latter assigns either zeroes or ones depending on the relevance of the vector.

3 MAMBA

In this section, we describe our method called Multi-Agent Model-Based Approach (MAMBA). First, we formulate the desiderata for the method and review existing papers that tackle the outlined problems. Second, we describe our method and its communication properties.

3.1 Desiderata

Sample Efficiency. One of the main strengths of MBRL approaches is the sample efficiency that results from using imaginary rollouts provided by the learned model. In the single-agent setting, multiple papers provide empirical evidence [23, 25, 33] as well as theoretical justifications [1, 4, 43] that MBRL approaches achieve near-optimal sample efficiency compared to Model Free approaches. This theory is further extended to the multi-agent setting [45, 79]. However, empirical performance is still lacking due to the increased complexity of the model and exponential growth of the action and state spaces.

Addressing non-stationarity. Experience replay buffer is a popular technique for off-policy learning that has been shown to improve sample efficiency compared to on-policy algorithms [2]. However, in multi-agent environments collected experience quickly becomes irrelevant due to the inherent non-stationarity [27]. Methods have been proposed by [19] to alleviate non-stationarity using techniques such as importance sampling and fingerprinting. However, these techniques can suffer from large variance as the agents progressively change their policies [33]. Another approach is to use on-policy algorithms and a world model that approximates the environment’s dynamics, which can be used to roll out imaginary trajectories starting from the states in the buffer for current policies, thus mitigating any issues with irrelevant experience [79].

Scalability. One of the key problems of learning dynamics of multi-agent environments is the curse of dimensionality. We want the world model to scale to a large number of agents in the environment, which would require the model to manage the resultant exponential growth of state and action spaces. This would allow it to process large numbers of transitions during both training and inference phases.

Locality. Multiple studies [46, 73] have shown that in large environments it is possible to significantly reduce the state space to comprise only relevant information for the agents’ decision making. This reduction can be naturally applied to environments like autonomous driving [63], where an agent only needs to be informed about cars in its local view. This property of locality can also be applied to communication [46, 80] by the same reasoning. Ideally, a world model should have a locality property, so that if the environment allows us to define neighbours of the agent, we want to communicate only with them during execution.
Discrete Communication. One dichotomy between the communication protocols is that the message can be either continuous [66] or discrete [20]. It is desirable that agents communicate using discrete messages, as it is more suitable for channels with limited bandwidth. Discrete representation allows us to greatly reduce the amount of information broadcast by the agents, which can be essential for deploying communication methods in the real world. Note that even with a restricted message channel, discrete messages can greatly enhance the performance of the agents [20].

Decentralized Execution. Finally, we want to deploy agents in a decentralized fashion, so they can make their decisions independently based on their local observations and messages from other agents [80]. To this end, one of the main paradigms is CTDE [18, 46, 54, 66], which achieves strong performance in multi-agent domains while maintaining decentralization on the execution stage. We want to learn a world model that can be sustained in a decentralized manner during execution using only communication between agents.

3.2 Architecture

We chose DreamerV2 [25], a state-of-the-art MBRL approach for discrete environments, as a backbone for our algorithm. Following Dreamer notation, the model consists of six components:

\[
\text{RSSM} = \begin{cases} 
\text{Recurrent model:} & h_t = f_\theta(h_{t-1}, z_{t-1}, a_{t-1}) \\
\text{Representation model:} & z_t = q_\phi(z_t | h_t, o_t) \\
\text{Transition predictor:} & \hat{z}_t = p_\nu(\hat{z}_t | h_t) \\
\text{Observation predictor:} & \hat{o}_t = p_\psi(\hat{o}_t | h_t, z_t) \\
\text{Reward predictor:} & \hat{r}_t = p_\psi(\hat{r}_t | h_t, z_t) \\
\text{Discount predictor:} & \hat{\gamma}_t = p_\psi(\hat{\gamma}_t | h_t, z_t) 
\end{cases}
\]

where the RSSM model [24] is used to learn the dynamics of the environment and Observation, Reward and Discount predictors are used for reconstructing corresponding variables. We learn Observation, Reward and Discount predictors similar to [25] via supervised loss and Representation model by minimizing the evidence lower bound [35]. Transition predictor is then trained by minimizing KL divergence between $\hat{z}_t$ and $z_t$, where $z_t$ is a stochastic state sampled from categorical distribution same as in DreamerV2. Recurrent model consists of 1 layer GRU [11] as in the original DreamerV2 implementation. In order to facilitate independence between hidden states $h_t$, we predict individual reward, discount and observation only from agent’s latent state. We also use the sum of KL divergences between individual stochastic states $\hat{z}_t$ by applying the chain rule to the true Transition function:

\[
p_\phi(\hat{z}_t | h_t) = p_\phi(\hat{z}_1 | h_1) \cdot p_\phi(\hat{z}_2 | h_2, \hat{z}_1) \cdots p_\phi(\hat{z}_t | h_t, \hat{z}_{t-1}, \ldots, \hat{z}_1)
\]

However, predicting latent states autoregressively is too impractical and we elaborate on its substitute in the next paragraph. The final loss for learning the world model consists of:

- Observation loss: $- \ln p_\psi(\hat{o}_t | h_t, z_t)$
- Reward loss: $- \ln p_\psi(\hat{r}_t | h_t, z_t)$
- Discount loss: $- \ln p_\psi(\hat{\gamma}_t | h_t, z_t)$
- KL divergence loss: $\mathrm{KL} \left[ q_\phi(z_t | h_t, o_t) \Vert p_\psi(\hat{z}_t | h_t) \right]$

which is calculated in expectation over $q_\phi(z_{1:T} | a_{1:T}, o_{1:T})$ and summed over all $n$ agents and rollout length $T$. As in Dreamer, all losses are optimized jointly using the Adam optimizer [34]. Note that we do not use an additional $\beta$ weight for KL divergence [28] to simplify algorithm complexity. As in DreamerV2, we use KL balancing to facilitate learning of prior $p_\psi(\hat{z}_t | h_t)$ by assigning a larger weight to cross entropy and a lower weight to posterior entropy in the KL term. The schematic representation of the training phase is in Figure 1.

Transition function. In our method, we implement Transition function using Attention. Simply predicting next latent states independently using Attention can create inconsistencies between agents’ current views of the environment, which can hinder performance. To solve this issue, we could theoretically predict next
latent states autoregressively, that is, condition the latent state of \( i \)-th agent on the predicted latent states of the previous agents. However, this would significantly slow down training and would not allow decentralized decision making during execution. Another approach, proposed by [39], is to maximize mutual information between the latent state and the previous action of the agent:

\[
\mathcal{L}_{\text{info}} = -I(h_t^i, z_t^i; a_{t-1}^i)
\]

(4)

This way, we encourage the latent state to depend mostly on its own actions, thus making it less dependent on others. Consequently, this disentanglement of the agent’s latent space allows us to dispatch them separately. We use the lower bound proposed by [10] to practically maximize mutual information by training an additional neural network to predict previous action from the latent state of the agent:

\[
\mathcal{L}_{\text{info}} = - \ln p_{\hat{\psi}}(a_{t-1}^i | h_t^i, z_t^i)
\]

(5)

**Stochastic state.** During the execution and imaginary rollout stages, agents are allowed to transmit only their stochastic states \( z_t^i \). As in DreamerV2, stochastic state has a discrete nature and consists of 32 categorical distributions with 32 classes each. Straight-through gradients [6] are used to learn these discrete representations. By design, stochastic state is the only message that can be broadcasted by the agent, which amounts only to \( 32 \log 32 = 160 \) bits of information per message. This is largely because of the sparsity of the \( z_t^i \) compared to the other possible distributions. We can also use a language-based interpretation [20, 42] of\( z_t^i \) as we restrict agents to a very limited vocabulary to express their current states. We further elaborate on this point in the next section.

**Communication Block.** We use Transformer architecture [71] with 3 stacked layers for encoding state-action pairs \((z_t, a_t)\). These encodings \( e_t \) are then used by the agents to update the world model and make action predictions. We use a dropout technique [65] with probability \( p = 0.1 \) to prevent the model from overfitting and use positional encoding [71] to distinguish agents in homogeneous settings.

**Absorbing states.** One major difference between single- and multi-agent environment dynamics is the varying number of agents on the map. Even if the starting number of agents is always the same, it is natural that agents will have varying trajectory lengths depending on the task. Thus, it is essential to handle the states of the agents that are no longer present in the environment during the training phase. Absorbing state [37] is an imaginary state that does not yield a reward for any action and does not lead to any other state, creating an infinite loop to itself. In our implementation, we add absorbing states after the end of an agents trajectory, so that it does not get a reward for future actions and so other agents can see that agent is no longer on the map. We do not use observation loss to learn the representation of those states as no meaningful observation can be provided by the environment. Instead, we allow the world model to implicitly learn those representations only from predicting the discounts and rewards that are both equal to zero.

**Behaviour learning.** We chose Actor-Critic framework to learn an agent’s behaviour, where both actor and critic are parameterized by neural networks \( \psi \) and \( \eta \) respectively:

\[
\text{Actor: } a_t^i \sim p_{\psi}(a_t^i | \hat{z}_t^i, h_t^i)\\
\text{Critic: } v_t^i(\xi_t) \approx \mathbb{E}_{p_{\nu}} \left[ \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} \right]
\]

*Imaginary rollouts.* As in Dreamer, we use learned transition function \( p_{\hat{\psi}}(\hat{z}_{t+1}^i | h_t^i) \) to predict the next stochastic state based on agents’ actions and current states of the model. Starting from initial states sampled from the replay buffer, we generate rollouts based on current policies of the agents in order to deploy on-policy learning algorithms in the latent space. We set the length of the imaginary rollouts to \( H = 15 \) to account for increased complexity of the multi-agent environments and possible compounding error [31] of the world model.

**Figure 2: Execution Phase.** Each agent has its own version of the world model, which can be updated using the Communication Block. It is sufficient to send only stochastic state \( z_{t-1}^i \) and action \( a_{t-1}^i \) from the previous step for each agent in order to obtain feature vectors \( e_t^i \). Agent \( i \) then can use the updated world model and its current observation \( o_t^i \) to output its next action \( a_t^i \).

**Actor.** Instead of using Reinforce [76] for policy updates as in Dreamer, we found that for multi-agent environments PPO updates (Eq. 1) yield better results. We do not use gradients backpropagated through the dynamics model as they hinder the performance, which is likely due to its high noisiness in multi-agent environments. We use parameter sharing [22] for the actor network to accelerate training.

**Critic.** We use \( \lambda \)-target [58] for value function as in Dreamer, which gives us more unbiased estimates than TD-error [69]. We augment critic by giving it as an input all current latent states \( \hat{z}_t \), which are processed using Attention mechanism. Unlike other CTDE approaches [47, 54], we cannot use additional information beyond the states predicted by the model as we train agents in the latent space. However, several studies [13, 74] suggested that such additional information can cause centralized-decentralized mismatch, which can hinder the learning process due to the large variance of the policy gradient updates. Furthermore, in real world
3.3 Communication

Here, we discuss the properties of our communication protocol. Similar to methods employing Attention mechanism to collect useful information from other agents [12, 32, 46], we allow agents to broadcast their latent state representations from the previous step \( z_{t-1} \). We also assume that agents can observe actions \( a_{t-1} \) of others. Tuples \((z_{t-1}, a_{t-1})\) are then transformed by Communication Block to feature vector \( \phi_t \) that is used to update the agent’s world model and predict next action based on the updated latent state. The schematic representation of the execution phase is in Figure 2.

**Reward-agnostic communication.** One important distinction of MAMBA communication from other communication protocols [12, 20, 66] is that it is not learned from the reward signal. We will refer to our communication protocol as reward-agnostic communication as we use a world model to obtain it, and we will denote communication protocols from [12, 20, 66] as goal-oriented communication as they are shaped by the agent’s reward. We argue that reward-agnostic communication is more suited for language that describes current environment while goal-oriented communication is more suited for describing agent’s task. This duality is mirrored in language theory as representation and acquisition theories respectively [17]. We can think of reward-agnostic communication as an emergent language that agents cannot alter to their needs but in exchange this language can provide an unbiased representation of the current state of the agent. On the other hand, goal-oriented communication is defined by the needs of the agent, thus making it a useful tool to achieve its goals. Consequently, many previous works on communication [8, 9, 51] studied this phenomenon as selfish agents tried to manipulate the communication channel to their own advantage by sending false messages to others. Conversely, we cannot use reward-agnostic communication in mixed environments as it provides true information about the agent to both allies and competitors.

**Locality.** As we want to preserve locality as much as possible, we can make the agent receive messages only from its neighbours in the environment. Attention mechanism allows us to vary the number of messages and, assuming that agents which are not neighbours will not have an impact on the agent transition function, it is possible to mask weights of those agents in Attention.

**Discrete communication.** Multiple studies have shown that discrete messages enhance performance as agents develop the language to communicate with each other [20, 26, 42]. On the other hand, language capabilities to succinctly describe images in only a few words have inspired several studies in representation learning [52, 55] to use discrete stochastic states. In our communication protocol, we can see the duality of these ideas as we can train stochastic states on unsupervised loss to reason about the current state of the agent and at the same time use it to communicate agent’s current surroundings.

**Computation-Communication trade-off.** Broadcasting only tuples \((z_{t-1}, a_{t-1})\) to all agents requires \(O(n^2)\) computations on each agent to calculate single-layer attention encodings [71]. Alternatively, agents can compute only one vector from \( n \), for example, \( qfK^T \) from Eq. 2, and exchange computed vectors on each step of attention to reduce computation complexity to \( O(n) \).

3.4 Limitations and Improvements

MAMBA satisfies all of the stated desiderata by leveraging a Model-Based approach for large scale multi-agent environments. However, MBRL has several limitations compared to MFRL methods. First, learning the world model introduces additional complexity to the Reinforcement Learning algorithm, which requires more careful tuning of hyperparameters. Second, while MAMBA is able to achieve good performance in most environments in a low data regime, it also requires more computational time for one collected trajectory compared to MFRL methods as it needs to learn the world model and rollout imaginary trajectories to learn the policy. We provide detailed comparisons of average wall clock time for training and execution with the same hardware in Appendix.

4 EXPERIMENTS

4.1 Environments

**Flatland.** [50] (Figure 3a) is a multi-agent 2d environment simulating train traffic on the rail network. Each agent has its own starting point and destination and the goal of the environment is to guide agents to their destinations without collisions. The difficulty of the environment can be flexibly adjusted as the number of agents as well as the size of the map could be easily configured. Note that maps are generated procedurally for each trajectory, so the world model must be able to generalize well in order to achieve good performance. Each agent receives positive reward on arrival
Figure 4: Experiments in SMAC environments. Y axis denotes winning rate of the agents and X axis denotes number of steps taken in the environment.

to its destination and negative reward if a collision occurs. We additionally use more dense variant of the reward provided by the winning solution of Flatland 2020 RL challenge [40]. Agent receives additional positive reward if its action shortens the approximated distance to its destination. This reward allows us to alleviate credit assignment issue and facilitates exploration for better performance. In this environment, the density of the traffic can be quite sparse, allowing us to test locality of our approach. During the execution phase, the agents can send messages only to their neighbours, which are defined by their proximity on the railroad.

**StarCraft Multi-Agent Challenge. (SMAC)** [56] (Figure 3b) is a suite of cooperative multi-agent environments based on StarCraft II. Each task consists of a single scenario with two confronting teams, one of which is controlled by the game bot and the other by our algorithm. The goal of the environment is therefore to defeat enemy team. SMAC offers 23 different scenarios with varying difficulty and number of agents. SMAC focuses on learning micromanagement of a group of agents, e.g. using terrain to your advantage as in corridor map or kite opponents as in 3s vs 3z map. Learning this behaviours requires both good exploration properties as well as solving credit assignment issue as agents need to disentangle global reward for their actions. One important property of the environment is that not all actions are available during agents decision making, e.g. attacking opponents that are out of reach. This necessitates world model to predict if the action is available or not based on latent state in order to train agents in imaginary rollouts. We use additional Bernoulli classifier similar to Discount predictor and train it on action masks provided by the environment for each agent.

4.2 Results

We adopt low data training regime from [33] to highlight sample efficiency of the current state-of-the-art MFRL methods. We chose MAPPO [78], COMA [18], QMIX [54], CommNet [66] and G2A [46] as the baselines for SMAC environments and winning solution of [40] as the baseline for Flatland environment. There are three levels of difficulty both in SMAC and Flatland. In order to make up for this difference, we allocate more samples from the environment depending on its difficulty. Specifically, we use 100k, 200k and 500k samples for SMAC and 300k, 1kk and 1.5kk for Flatland. All hyper-parameters, ablation study and details about the environments can be found in Appendix. We use this implementation\(^2\) for all baselines in SMAC except MAPPO, which implementation is provided by authors\(^3\). We use authors implementation of the winning Flatland 2020 solution that can be found here\(^4\).

SMAC. The results in SMAC environments are presented in Figure 4. We selected 8 environments out of 23 to show in the paper and the rest of the results can be found in Appendix. Specifically, we chose 4 random environments (Fig. 4 a, b, c, d) that are considered easy tasks, 2 environments (Fig. 4 e, f) with large number of agents and 2 environments (Fig. 4 g, h) with very hard tasks that require coordination. SMAC environments require coordination between all agents to complete the task, thus we did not test the locality property of our algorithm here. We can see that MAMBA outperforms all baselines in all of the environments with given number of samples. Notably, MAMBA achieves near-optimal performance on easy tasks with only 100k samples and finds winning strategy for hard and very hard tasks, winning nearly half of the episodes. On

\(^2\)https://github.com/starry-sky6688/StarCraft
\(^3\)https://github.com/marlbenchmark/on-policy
\(^4\)https://github.com/jbr-ai-labs/NeurIPS2020-Flatland-Competition-Solution
the other hand, the current state-of-the-art MFRL method MAPPO, which is able to solve all SMAC environments given large amount of samples, struggles to achieve good performance in low data regime. This highlights sample inefficiency of current MFRL approaches and shows the potential for a significant reduction in needed samples using MBRL methods. Note that communication methods G2A and CommNet are less sample efficient than MAPPO, which is likely due to additional complexity of learning how to leverage message channel. On the other hand, MAMBA encapsulates communication in a world model, which can be learned in fewer samples. Two important highlights of SMAC environments are the credit assignment problem and usage of additional information during training. We can see that MAMBA is able to disentangle global rewards and achieve good performance even without additional credit assignment techniques in most environments. We do not use additional global information during training, which is a limitation of model-based approaches discussed in Section 3.2.

**Flatland.** The results in Flatland environment are presented in Figure 5. We use modified version of the winning solution from the RL track of Flatland 2020 challenge [40] as our main baseline, which we will refer to as 1RL. Our only modification is disabling departure schedule, as it alters the underlying MDP of the environment. As there is no global reward in the environment, we do not compare with algorithms specific for solving credit assignment issue, e.g. COMA, QMIX and QTRAN. We also found it redundant to compare with MADDPG or communication methods such as G2A or CommNet as the winning solution employs PPO-based method with both augmented critics and communication between agents. In this environment, we use three different maps with the increasing size and number of agents. That way, we can progressively increase the difficulty of the environment and see how this affects the algorithms. Specifically, we use environments with 5, 10 and 15 agents, which turn traffic on the rail network more and more dense. This increase in difficulty affects the learning of world model as it must predict next states of all agents during training. Furthermore, during execution we provide agent messages only from its neighbours for both 1RL and MAMBA, thus limiting information about the environment only to local area of the map. Nevertheless, we can see that MAMBA can successfully achieve good performance in the 5 agents (Fig. 5 a) setting only in 300k samples while 1RL still needs more samples to improve performance. Naturally, performance decrease in 10 agents setting for both algorithms, but we can see that MAMBA still substantially outperforms 1RL method in 1 million samples (Fig. 5 b). However, we can see that sample efficiency starts to equalize compared to 5 agents setting. We hypothesize that it is easier for 1RL agents to learn on raw observations with communication than for MAMBA agents to learn from more noisy discrete latent space of the model. This issue of decaying sample efficiency is further exacerbated with the increased number of agents in the next setting. In 15 agents setting (Fig. 5 c) we can see that MAMBA is less sample efficient than 1RL but still achieves comparable performance in 1.5 million samples. We hypothesize that this decrease in sample efficiency results from the increased difficulty of the environment, which necessitates more expressive world model to approximate as well as more samples to learn.

5 CONCLUSION

In this paper, we present MAMBA – a MBRL method for multi-agent cooperative environments. We show the effectiveness of MAMBA in low data regime on two challenging domains of SMAC and Flatland compared to current state-of-the-art approaches, highlighting sample inefficiency of the MFRL methods. We provide architecture that can scale to a large number of agents as well as a training procedure that can learn disentangled latent space for agents. This allows us to decompose the world model by the agent, which can then be used in a decentralized manner with communication. Furthermore, we use discrete messages to facilitate decentralization and account for message channel bandwidth limitations.

6 ACKNOWLEDGEMENTS

The publication was supported by the grant for research centers in the field of AI provided by the Analytical Center for the Government of the Russian Federation (ACRF) in accordance with the agreement on the provision of subsidies (identifier of the agreement 000000D730321P5Q0002) and the agreement with HSE University No. 70-2021-00139. This research was supported in part through computational resources of HPC facilities at HSE University [36].
