Model-based Sparse Communication in Multi-agent Reinforcement Learning

Shuai Han
Utrecht University
Utrecht, the Netherland
s.han@uu.nl

Mehdi Dastani
Utrecht University
Utrecht, the Netherland
m.m.dastani@uu.nl

Shihan Wang
Utrecht University
Utrecht, the Netherland
s.wang2@uu.nl

ABSTRACT

Learning to communicate efficiently is central to multi-agent reinforcement learning (MARL). Existing methods often require agents to exchange messages intensively, which abuses communication channels and leads to high communication overhead. Only a few methods target on learning sparse communication, but they allow limited information to be shared, which affects the efficiency of policy learning. In this work, we propose model-based communication (MBC), a learning framework with a decentralized communication scheduling process. The MBC framework enables multiple agents to make decisions with sparse communication. In particular, the MBC framework introduces a model-based message estimator to estimate the up-to-date global messages using past local data. A decentralized message scheduling mechanism is also proposed to determine whether a message shall be sent based on the estimation. We evaluated our method in a variety of mixed cooperative-competitive environments. The experiment results show that the MBC method shows better performance and lower channel overhead than the state-of-art baselines.

KEYWORDS

Multi-Agent Reinforcement Learning; Multi-Agent System; Communication Learning; Message Scheduling

ACM Reference Format:
Shuai Han, Mehdi Dastani, and Shihan Wang. 2023. Model-based Sparse Communication in Multi-agent Reinforcement Learning. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

1 INTRODUCTION

Multi-agent reinforcement learning (MARL) provides powerful approaches for agents to develop effective cooperative and competitive policies. Recently these approaches have been applied in a variety of complex environments, such as traffic light control [37], robotics [9] and autonomous driving [36]. Communication allows agents to share observations and intentions, thus greatly improving the efficiency and success rate for completing specific tasks [18, 23, 32]. In communication MARL, agents communicate with each other before taking action and learn message encoding and decoding protocol with backpropagation [33]. In addition to encoding and decoding messages, agents need to learn 'when' and 'whom' to send their messages. This is known as communication scheduling [15, 23]. For example, IC3Net [32] learns when to broadcast messages, and I2C [4] and ACML [20] introduce gate mechanisms to decide whether to send messages to specific agents. Moreover, FlowComm [5] and MAGIC [23] learn to dynamically generate communication graphs to schedule messages.

However, most communication MARL methods require intensive communication among agents, which leads to high communication overhead. This issue is crucial for the real-world application of MARL methods where communication is costly [39]. For example, in some practical scenarios, such as unmanned aerial vehicles, reducing communication overhead is a fundamental concern due to the low-power property of the sensors [10]. The existing communication MARL algorithms rarely consider the cost during training and execution, resulting in excessive and redundant communication [5, 40, 42]. On the other hand, a few existing methods for reducing communication overhead [15, 18, 42] do not allow agents to utilize enough information about the observations and intentions of other agents, resulting in uncompetitive performance. To our knowledge, reducing communication overhead while enabling agents to use as much information as possible to learn optimal policies is a problem that has rarely been studied.

In order to deal with this problem, we propose a novel framework in communication MARL, which we will call model-based communication (MBC). The basic idea of MBC is to enable agents to utilize the previously exchanged messages to estimate current messages that agents may exchange. The estimated messages, accessible to all individual agents, can be used by individual agents to decide whether it is needed to send a message (in case the message deviates significantly from the estimated message) or if the other agents can use their estimated message. The latter case will reduce communication overhead.

In the MBC framework, this is realized by a message estimator that is designed to be trained in a supervised manner to model the dynamics of global messages, so that agents can estimate the current messages of other agents using their previous messages. In addition, a decentralized message scheduling mechanism is introduced, which eliminates the necessity of additional communication to a central scheduler and is therefore conducive to distributed deployment. According to the scheduling in the MBC framework, each agent will send its messages only when other agents cannot estimate its messages within an error threshold. In this way, MARL agents can correct the estimation error of the global message by sending real messages with each other.

This paper makes the following contributions. 1) We propose to reduce the communication overhead by replacing receiving messages from other agents with estimating others' messages. To

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

the best of our knowledge, this is the first work to use a model-based method to reduce the communication overhead in MARL approaches. 2) We design a decentralized message scheduling mechanism for multi-agents to correct erroneous message estimations by communicating the real messages when they cannot estimate the message accurately. 3) We verified the performance of our approach in a series of mixed cooperative-competitive environments. Our approach shows around 5.16% better performance on average than the state-of-the-art methods, around 22.33% lower communication overhead on average than the previous most communication-efficient method, and less dependence on the number of channels.

2 RELATED WORK

MARL is dedicated to learning joint policies capable of accomplishing specific tasks in dynamic and complex environments. Among the plethora of work on MARL [8, 31, 44], we summarize the literature in three subfields, which our approach either builds on or closely relates to.

Communication scheduling in MARL. This subfield addresses the problems of when to communicate and whom to address messages to. As the early works, IC3Net [32] and SchedNet [15] learn to decide when to broadcast individual messages. In other approaches based on the importance of individual messages to the system decisions, I2C [4] and ACML [20] introduce gate mechanisms to decide whether to send messages for individual agents. Graphs are also useful utilities for portraying the communication relationships among agents. Agent-Entity Graph [1] and G2ANet [18] schedule communication among agents via a pretrained graph. Most recently, FlowComm [5] and MAGIC [23] learn to dynamically generate graphs to determine when to communicate with whom. Communication scheduling can reduce the communication overhead in multi-agent systems, which is significant for deployment of MARL for real-world scenarios [18, 23, 42].

In line with these works, our scheduling approach further reduces communication load while maintaining collaboration among agents. Our method is different from these methods in terms of the concepts used by the message scheduling mechanism. The above methods schedule messages based on the impact of individual information on the reward function or on the decisions of other agents [15, 20, 23]. Our approach, in contrast, schedules messages to control the errors of individual estimations on global information. Methods that schedule messages based on message errors also include VBC [41] and TMC [42]. However, with the sparse communication of these methods, there are very few messages that agents can utilize. Unlike these methods, our approach enables agents to make estimations on the current global messages and makes decisions based on these estimated global messages.

Message aggregation in MARL. The approaches in this area address how to learn to effectively extract information from the received messages. Earlier approaches mostly average [32, 33, 41] or concatenate [15] the received messages to aggregate them. This linear aggregation approach works but cannot differentiate valuable information that helps decision making when there is a large number of agents. Some practices, such as pruning the incoming messages [5, 18] or adding attention mechanisms to them [3, 14], can alleviate this problem. A recent popular approach is to use the powerful representation learning capability of Graph Neural Networks (GNNs) to learn embeddings from system communication topology [23, 29, 43]. Our approach builds on and adopts the multi-layer nonlinear aggregation methods on received messages using the Graph Attention Networks (GATs) [2, 23] or attention mechanisms [3]. In particular, to overcome the problem of numerous input messages, our approach uses GATs to aggregate the information from estimated global messages.

Model-based multi-agent reinforcement learning. Model-based MARL alleviates the issue of sample efficiency in model-free MARL [6, 38]. These methods typically model the environment through supervised training. After the model learns to accurately simulate the environment, the agent interacts with this model rather than the environment [38]. H-MARL uses models to improve performance by considering equilibrium policies at each decision [30]. MAMBA applies the centralized training & decentralized execution paradigm to model-based multi-agent reinforcement learning [6]. Inspired by the previous Model-based MARL approaches, we propose to train a model in a supervised way to estimate the environmental dynamics. In particular, instead of modelling the dynamics of the observations (in previous works), we model the dynamics of communication messages (i.e. encoded observations). To the best of our knowledge, this is the first work to introduce the model-based learning into communication of MARL.

3 PRELIMINARIES

3.1 Markov Game

We follow the partially observable multi-agent Markov Game [17, 23] to study the communication in multi-agent reinforcement learning. A partially observable multi-agent Markov Game is defined as a tuple $< \mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, R, \gamma >$. In this tuple, $\mathcal{N} = \{1, 2, ..., n\}$ is the set of agents, $\mathcal{S}$ is the set of global state, $\mathcal{A} = A_1 \times A_2 \times ... \times A_N$ is the set of joint actions where A_i is the set of possible actions of agent i, $\mathcal{T} : \mathcal{S} \times A_1 \times ... \times A_N \mapsto \mathcal{S}$ is the transition function, $\Omega = \Omega_1 \times \Omega_2 \times ... \times \Omega_n$ is the set of joint observations where Ω_i is the possible observations of agent i, R is the reward function, and $\gamma \in [0, 1]$ is a discounted factor. At each time step t, agent i executes an action $a_i^t \in A_i$ based on its observation $o_i^t \in \Omega_i$, and receives an individual reward r_i. Agent i is dedicated to adjust its policy π_i to maximize the individual rewards: $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$, where T is the terminal time step.

3.2 MARL with Communication

When the above Markov game has been solved using multi-agent reinforcement learning with communication, agent i makes decisions based on individual policy $\pi_i : \Omega_i \times AM_i \times A_i \mapsto [0, 1]$, where AM_i is the set of aggregated messages received by the i-th agent. Individual agent i uses $f_{agg_i} : M_1, M_2, ..., M_N \mapsto AM_i$ to aggregate received messages am_i where M_j represents the set of agent j's possible sending messages. Moreover, agent j who sends message uses $f_{enc_j} : \Omega_j \mapsto M_j$ to encode local observation into message m_j. π_i, f_{agg_i} and f_{enc_j} are jointly parameterized by $\theta_{i,j} = [\theta^{\pi_i}, \theta^{f_{agg_i}}, \theta^{f_{enc_j}}]$, where θ^{π_i} is the parameters of π_i, $\theta^{f_{agg_i}}$ is the parameters of f_{agg_i}, and $\theta^{f_{enc_j}}$ is the parameters of f_{enc_j}.

In this work, we follow the previous policy gradient methods [21, 23, 28] to train policy π_i, f_{agg_i} and f_{enc_j} jointly by maximizing the objective $J(\theta_{i,j}) = \mathbb{E}_{s\sim\rho,a_i\sim\pi_i}[R_i^t]$, where ρ is the initial state distribution and $R_i^t = \sum_{t'=t}^{T} \gamma^{t'-t} r_i^{t'}$ is the discounted total rewards of agent i from the time step $t \in [0, T]$. This method performs gradient ascent on $\theta_{i,j}$.

$$\nabla_{\theta_{i,j}} J(\theta_{i,j}) = \mathbb{E}_{s\sim\rho,a_i\sim\pi_i}[\sum_{t=1}^{T} \nabla_{\theta_{i,j}} \log \pi(a_i^t|o_i^t, am_i^t) \cdot R_i^t] \quad (1)$$

where $am_i^t = f_{agg_i}(m_1, ..., m_j, ..., m_N)$ and $m_j = f_{enc_j}(o_j^t)$. To reduce the variance, we adopt the advantage function $A_i(o_i^t, a_i^t) = R_i^t - V_i(o_i^t)$ in place of R_i^t, where V_i is the expected cumulative reward estimated by agent i.

3.3 Centralized Training & Decentralized Execution.

Centralized training & decentralized execution (CTDE) is a common paradigm in MARL [13, 25, 34]. In this paradigm, all global information is available in training, but in execution, agents are only allowed to utilize local or received information. The existing approaches mostly use centralized value function and decentralized policies [7, 26, 27]. When it comes to communication, the encoding and decoding of messages are performed decentralized in execution [4, 32, 33]. Message scheduling has both decentralized [42] and centralized methods [23] in execution. When a centralized message scheduler exists in the system, it is often necessary to collect messages from all agents first before scheduling.

Our approach follows the CTDE paradigm. During training, we centrally train the model for estimating message dynamics in a supervised manner. During execution the scheduling and aggregation of messages are both executed in a decentralized manner.

4 METHODOLOGY

In this section, we describe the detailed design of the Model-based Communication (MBC) framework. The core idea of MBC is to estimate the current global message using local historical information to reduce the requirement of receiving messages directly from other agents. This enables each agent to maintain an error-controlled overview of the global information under a partial communication setup, thus achieving high performance with low communication overhead. We first introduce the overview structure of the MBC framework, then present the detailed design of each module.

4.1 Overview of the MBC Framework

We start with an intuitive example about how an individual agent makes decisions with communication (as shown in Figure 1).

To simplify the description, we demonstrate the procedure in a four-agent setting. At time step $t-1$, agent 1 uses ***Decision Generator*** to take action a_1^{t-1} based on the local encoding m_1^{t-1} as well as the aggregated message am_1^{t-1}. The m_1^{t-1} comes from its ***Message Encoders*** by encoding the local observation o_1^{t-1}. The am_1^{t-1} is computed by the ***Message Aggregator*** using the overwritten message vector $[(\hat{m}_1^{t-1})_1, (\hat{m}_2^{t-1})_1, m_3^{t-1}, (\hat{m}_4^{t-1})_1]$. The components of this vector are either estimated messages computed by local agent 1 (i.e., $(\hat{m}_1^{t-1})_1$, $(\hat{m}_2^{t-1})_1$, $(\hat{m}_4^{t-1})_1$), or the real messages received from other agents (i.e., m_3^{t-1}).

To obtain this overwritten message vector at $t-1$, agent i first estimates the current global message vector $[(\hat{m}_1^{t-1})_1, (\hat{m}_2^{t-1})_1, (\hat{m}_3^{t-1})_1, (\hat{m}_4^{t-1})_1]$ via ***Message Estimator*** based on the overwritten message vector provided from the previous time. Then, the received real message will overwrite this estimated message vector to reduce the error of estimation. In this example, only $(\hat{m}_3^{t-1})_1$ is overwritten, because agent 1 only receives messages m_3^{t-1} from agent 3 at this moment.

In addition, once Message Estimator obtains the estimated global message vector, the self-estimation component $(\hat{m}_1^{t-1})_1$ will be taken out. The scheduling process then compares estimated $(\hat{m}_1^{t-1})_1$ and real local message m_1^{t-1} to decide whether to send m_1^{t-1} to other agents. This sending process, together with the overwriting, constitutes the communication before the decision.

At the next time step, agent 1 will repeat the above process. Notably, the demonstrated process in Figure 1 works on each individual agent. In other words, all agents contain the same framework, and each one holds its individual modules (i.e., Message Estimator, Message Encoder, Message Aggregator, and Decision Generator) and performs their individual scheduling and overwriting processes. The parameters of the four modules are shared by all agents in the system. Next, we present more details about how various modules and corresponding processes are modeled in the MBC framework.

4.2 Decision-making with Messages

This subsection describes in detail how the Message Encoder encodes messages and how the Decision Generator generates actions. Figure 2 shows the structures of Message Encoder and Decision Generator. Agents in the system use a shared Message Encoder f_{enc} to encode the observations into messages. Specifically, the observation o_i^t of any agent i is initially encoded by a fully connected layer FC1, and then further encoded together with collected information from previous steps by a Long Short-term Memory (LSTM) layer [11] into m_i^t.

Thereafter, agents use a shared individual π to make decisions based on the encoded observations and aggregated messages. Specifically, the encoded observation m_i^t is concatenated with the aggregated message am_i^t. Then, they are input into another fully connected layer FC2 to generate individual decision a_i^t. In summary, agent i generates an action through the following equation.

$$a_i^t = f_{FC2}(m_i^t || am_i^t) \quad (2)$$

where f_{FC2} represents the fully connected layer $FC2$ and $(\cdot||\cdot)$ represents the 'concatenation' operation shown in Figure 2.

4.3 Estimating Global Messages

This subsection describes how the Message Estimator is centrally trained and how the Message Estimator uses previous global message vector to estimate the current global messages.

Message Estimator builds up a supervised learning model to estimate the up-to-date global message information based on messages from the previous step. Since the key to our idea is to reduce the communication channel overhead by using estimated messages as

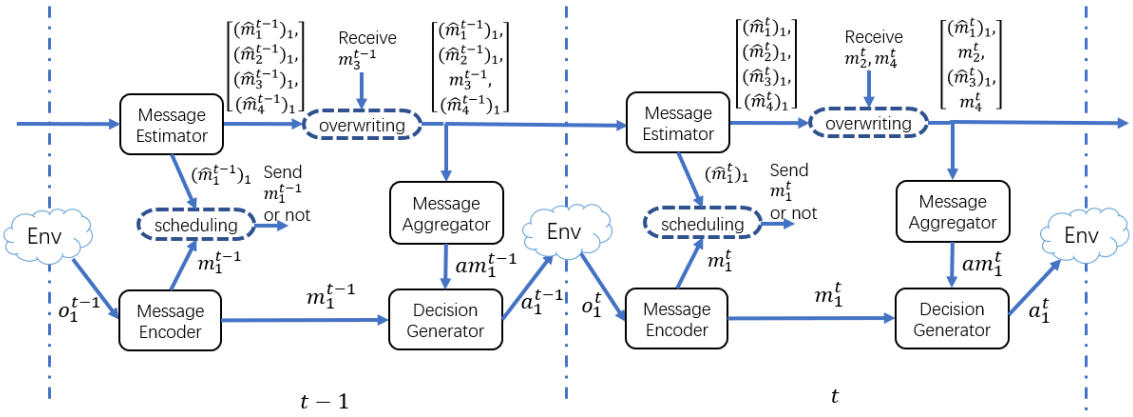


Figure 1: The demonstration of the proposed MBC framework, including the decision making process of one agent at time step t-1 and t in a four-agent MARL system. In addition to the notations mentioned in Section 3, $(\hat{m}_j^t)_i$ represents agent j's message at t estimated by agent i.

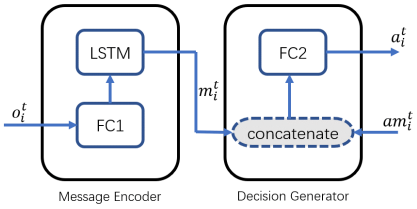


Figure 2: Network Structures of Message Encoder and Decision Generator. The 'concatenation' represents connecting two vectors together.

much as possible instead of received messages, training an accurate Message Estimator is essential.

To train an accurate Message Estimator, we seek inspiration from model-based reinforcement learning methods [12, 22]. In the model-based MARL methods, agents model the dynamics of the observations. This dynamics, which determines the joint observation $\boldsymbol{o^t} = [o_1^t, o_2^t, ..., o_N^t]$ at time t, is modelled by the to be learned environment model f_o:

$$\hat{\boldsymbol{o}}^{\boldsymbol{t}} = f_o(\hat{\boldsymbol{o}}^{\boldsymbol{t-1}}, \boldsymbol{a^{t-1}}) \tag{3}$$

where $\boldsymbol{a^{t-1}} = [a_1^{t-1}, a_2^{t-1}, ..., a_N^{t-1}]$ is the action profile consisting of the actions of all agents at time step $t-1$. Since $\boldsymbol{m^t}$ is essentially the encoding of $\boldsymbol{o^t}$, based on Equation (3), we assume that the dynamics of the messages, which determines the joint message $\boldsymbol{m^t} = [m_1^t, m_2^t, ..., m_N^t]$ at time t, can also be estimated by the learned message model f_M.

$$\hat{\boldsymbol{m}}^{\boldsymbol{t}} = f_M(\boldsymbol{m^{t-1}}, \boldsymbol{a^{t-1}}) \tag{4}$$

where $\hat{\boldsymbol{m}}^{\boldsymbol{t}}$ is the estimated joint message. To obtain the joint action $\boldsymbol{a^{t-1}}$, we calculate its components separately with Equation (2), $a_i^{t-1} = f_{FC2}(m_i^{t-1}||am_i^{t-1})$, where $am_i^{t-1} = f_{agg}(\boldsymbol{m^{t-1}})$ and f_{agg} is the function of Message Aggregator. Therefore, since $\boldsymbol{a^{t-1}}$ depends on $\boldsymbol{m^{t-1}}$, we only require $\boldsymbol{m^{t-1}}$ to calculate $\hat{\boldsymbol{m}}^{\boldsymbol{t}}$ in equation (4).

In this way, we learn a model for the global dynamic in Equation (4) instead of on individual dynamic in [16, 22]. Because in the MBC framework, an (overwritten) global message vector is always available for every agent, we can learn a global model to utilize more information as well as to train a stable message model.

During training, we collect real global messages and actions to train f_M in a supervised way. We use the Mean Square Error (MSE) between the true messages and the estimated messages as the loss function.

$$\mathcal{L}_M(\theta^{f_M}) = \mathbb{E}_{(\boldsymbol{m}^{t-1}, \boldsymbol{m}^t)\sim\mathcal{D}}[\boldsymbol{m}^t - f_M(\boldsymbol{m^{t-1}}, \boldsymbol{a^{t-1}})]^2 \tag{5}$$

where θ^{f_M} is the parameter of f_M and $\mathcal{D}$ is a buffer that stores tuples $(\boldsymbol{m}^{t-1}, \boldsymbol{m}^t)$ for $t = 1, 2, ..., T$. To better train f_M, we use $\mathcal{D}$ similar to [12, 22] to collect data as the training set during the interaction of agents with the environment. During training, $(\boldsymbol{m}^{t-1}, \boldsymbol{m}^t)$ will be sampled in batches from $\mathcal{D}$ to compute the gradient of Equation 5 and perform parameter updates on f_M.

In execution, we deploy the same parameters of the trained f_M to each agent locally. However, since the global information in Equations (4) and (2) is not always available for agent i, we use locally available information to estimate global message. Considering the general situation of Figure 1, agent i computes its own estimation on global message $\boldsymbol{(\hat{m}^t)_i} = [(\hat{m}_1^t)_i, (\hat{m}_2^t)_i, ..., (\hat{m}_N^t)_i]$ with the overwritten message vector $\boldsymbol{(\dot{m}^{t-1})_i} = [(\dot{m}_1^{t-1})_i, (\dot{m}_2^{t-1})_i, \ldots, (\dot{m}_N^{t-1})_i]$ from the last time step, where $(\dot{m}_j^{t-1})_i = (\hat{m}_j^{t-1})_i$ *or* m_j^{t-1} depending on whether agent i has received a real message m_j^{t-1} from j. If a message m_j^{t-1} was received, it will be used to overwrite the estimation $(\hat{m}_j^{t-1})_i$.

Therefore, in execution, agent i obtains its estimation on global message with the following equation.

$$\boldsymbol{(\hat{m}^t)_i} = f_M\left(\boldsymbol{(\dot{m}^{t-1})_i}, \boldsymbol{(\hat{a}^{t-1})_i}\right) \tag{6}$$

where $\boldsymbol{(\hat{a}^{t-1})_i} = [(\hat{a}_j^{t-1})_i]$ is the joint action estimated by agent i and $(\hat{a}_j^{t-1})_i = f_{FC2}((\dot{m}_j^{t-1})_i||f_{agg}(\boldsymbol{(\dot{m}^{t-1})_i}))$. Agent i needs to estimate the joint action because it cannot observe the global action at current time step in decentralized execution. Here, agent i is able to estimate the action of agent j individually because in our framework, different agents maintain similar overwritten global message, i.e., $\boldsymbol{(\dot{m}^{t-1})_j} \approx \boldsymbol{(\dot{m}^{t-1})_i}$, and all agents share the same

parameters of f_{FC2} and f_{agg}. The more components of $(\dot{\boldsymbol{m}}^{\boldsymbol{t-1}})_{\boldsymbol{i}}$ are overwritten, the more accurate this approximation is. This also means a higher communication overhead as more messages are received.

4.4 Scheduling Messages

This subsection describes how to schedule messages. The scheduling process determines whether the current local message should be sent to all other agents.

The local message scheduling aims to control the estimation error within a certain boundary. This can be done by checking whether other agents are able to properly estimate the local message of agent i at this moment. In other words, if other agents can estimate the message sent by agent i at time step t within the error bound, agent i does not need to send its local encoded observation m_i^t. (Message Estimator is shared and accessible to all agents in execution). Otherwise, m_i^t needs to be sent to other agents to help them recover from the incorrect estimation.

We propose to schedule the message in a decentralized way. This allows the message dispatching module to be better deployed in a distributed manner. To do this, it is necessary to use only the local information available during the execution. As mentioned in Subsection 4.3, each agent estimates others' current messages using Equation (6) during execution. Since each agent shares the same parameters of f_M, f_{FC2}, as well as f_{agg} and maintains similar global message vector inputs, we can use agent i's estimation on itself to approximate other agents' estimation on i, i.e., $(\hat{m}_i^t)_i \approx (\hat{m}_i^t)_j$, where $j \neq i$. In other words, if agent i cannot estimate its own current message m_i^t within an error range, agent i will assume that other agents cannot estimate m_i^t neither. Then, agent i should send its message to other agents to help other agents recover from a wrong estimation.

In summary, we design the message scheduling as the following equation.

$$I_i^t = \begin{cases} 1 & \text{if } ||(\hat{m}_i^t)_i - m_i^t||_2 > \delta, \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where I_i^t is a binary value that indicates whether agent i sends its messages to other agents at time step t, $||\cdot||_2$ indicates $L2$ norm of a vector, δ is a hyperparameter representing the threshold value.

In addition to the scheduling approach in Equation (7), to prevent the cumulative error in message estimation, we force agents to send messages if they have not sent a message after a time window w. Besides, when $t = 0$, we set all agents to send their messages to start the calculation of Equation (6).

4.5 Aggregating Messages

This subsection describes how to aggregate messages from the current overwritten global message vector.

Since each agent needs to aggregate useful information from N messages, we solve this high input dimensionality by following the ideas of GATs [2, 23]. In the calculation of individual agent i, the message in layer l is computed recursively by aggregating the messages from the last layer $l-1$:

$$(m_j^{t(l)})_i = \sigma\Big(\sum_{k\in\mathcal{N}} (\alpha_{jk})_i W^{(l)} (m_k^{t(l-1)})_i\Big) \tag{8}$$

where $(m_j^{t(l)})_i$ is the intermediate message component on layer l computed by agent i locally at time step t, $\sigma(\cdot)$ is a nonlinear activation function (i.e. LeakyReLU [24] in our experiments), $\mathcal{N}$ is the set of all agents in the system, $W^{(l)}$ is a learnable weighting matrix shared among all nodes at layer l, and $(\alpha_{jk})_i$ is the weight coefficient of agent k's message for agent j, as viewed by agent i. $(\alpha_{ij})_p$ is computed by the attention mechanism [2, 23].

From Equation (8), it can be seen that the f_{agg} aggregates messages recursively and that the parameters of the f_{agg} consist of the learnable $W^{(l)}$ at each layer. In L-layer GATs, the input of GATs first layer is $(m_j^{t(0)})_i = (\dot{m}_j^t)_i$, where $(\dot{m}_j^t)_i$ comes from the component of the overwritten message vector $(\dot{\boldsymbol{m}}^{\boldsymbol{t-1}})_{\boldsymbol{i}}$. The output of the final layer is the aggregated message of agent i: $am_i = (m_i^{t(L)})_i$.

Equation (8) differs from the previous work in that agent i performs the computation of all agents' intermediate aggregated messages locally and aggregates higher level aggregated messages from the local computation. This means that multi-layer message aggregation no longer requires multi-round communication in our approach, which is required in previous work [3, 23]. This allows the communication overhead of our method to be further reduced.

4.6 Algorithms

In this subsection, we describe the implementation of MBC training and execution by presenting our algorithms.

The training of Message Estimator is separate from the training of the other three modules. We use Equation (5) to train the Message Estimator and we use the objective $J(\theta_{i,j})$ in Subsection 3.2 to jointly train the other modules in MBC. In particular, due to the parameter sharing in MBC, the objective is presented as following.

$$\nabla_{\theta,\phi} J(\theta,\phi) =$$
$$\frac{1}{T}\sum_{i=1}^{N}\sum_{t=1}^{T}\left[\nabla_\theta \log \pi(a_i^t|o_i^t, am_i^t)(R_i^t - V_\phi(o_i^t)) - \beta\nabla_\phi(R_i^t - V_\phi(o_i^t))^2)\right] \tag{9}$$

where $am_i^t = f_{agg}(\dot{\boldsymbol{m}}^{\boldsymbol{t}})$, β is a weighting factor that determines the update weight of θ and ϕ, and V_ϕ is the value function parameterized by ϕ.

We present the training procedure of our proposed algorithm in Algorithm 1. We start with initializing the total number of updates M, max episode steps T, parameters for agents and buffer $\mathcal{D}$. Then the system agents interact with the environment in Step 8~9 and training data is collected and sorted in Step 12, 15, and 17. Finally, the parameters are update according the gradient in Step 19 and 20.

After training, the shared parameters θ, ϕ, and θ^{f_M} are deployed to individual agents for decentralized execution. The decentralized execution procedure for every individual agent within an episode is presented in Algorithm 2. The environment is randomly reset in the beginning of the episode and the initial observation o_i^0 of the individual agent i is obtained in Step 1. Then the i's estimation message vector and the window size is initialized in Step 2 and 3. Thereafter, t_{notc} and t are initialized as zero, where t_{notc} records the cumulative time steps during which agent i do not sent its message. Step 5 ~ 22 describes the decision process of agent i. The current observation o_i^t is first encoded into message m_i^t in Step 6, corresponding to Subsection 4.2. Then the message is scheduled

Algorithm 1 Centralized training for all agents.

Initialize max updates M, max episode time steps T
Initialize shared parameters θ and ϕ
Initialize shared parameters θ^{f_M} and buffer $\mathcal{D}$
for $update = 1 \rightarrow M$ **do**
  Reset Env and obtain o_i^0 for each agent i
  $t = 0$
  while $t < T$ and not terminal **do**
    Input o_i^t to each agent and obtain m_i^t and a_i^t
    Execute a_i^t to the Env and observe o_i^{t+1}, r_i^t
    Merge all m_i^t into $\boldsymbol{m^t}$
    if $t > 0$ **then**
      Store $(\boldsymbol{m^{t-1}}, \boldsymbol{m^t})$ into $\mathcal{D}$
    end if
  end while
  $R_i^t = 0$ if t is terminal else $R_i^t = V(o_i^t)$
  for $t = (T-1) \rightarrow 0$ **do**
    $R_i^t \leftarrow r_i^t + \gamma R_i^t$
  end for
  Calculate the gradient of the objective according to Equation (9) and update θ and ϕ
  Sample from $\mathcal{D}$ and calculate the gradient of the loss according to Equation 5 and update θ^{f_M}
end for

Algorithm 2 Decentralized execution for agent i in an episode.

1: Reset Env and obtain o_i^0
2: Initialize the message vector $(\hat{\boldsymbol{m}}^{\boldsymbol{0}})_{\boldsymbol{i}}$ estimated by i
3: Initialize the window size w
4: $t_{notc} \leftarrow 0, t \leftarrow 0$
5: **while** $t < T$ and not terminal **do**
6: Encode o_i^t as message m_i^t
7: **if** $t == 0$ or $t_{notc} > w$ **then**
8: Send m_i^t to all agents and $t_{notc} \leftarrow 0$
9: **else**
10: Estimate $(\hat{\boldsymbol{m}}^{\boldsymbol{t}})_{\boldsymbol{i}}$ based on $(\hat{\boldsymbol{m}}^{\boldsymbol{t-1}})_{\boldsymbol{i}}$ with Equation (6)
11: Obtain I_i^t based on $(\hat{m}_i^t)_i$ and m_i^t with Equation (7)
12: **if** $I_i^t > 0$ **then**
13: Send m_i^t to all agents and $t_{notc} \leftarrow 0$
14: **else**
15: $t_{notc} \leftarrow t_{notc} + 1$
16: **end if**
17: **end if**
18: Receive $\{m_j^t\}$
19: Overwrite the corresponding component of $(\hat{\boldsymbol{m}}^{\boldsymbol{t}})_{\boldsymbol{i}}$ with real messages $\{m_j^t\}$ and obtain $(\dot{\boldsymbol{m}}^{\boldsymbol{t}})_{\boldsymbol{i}}$
20: Compute $am_i^t = f_{agg}((\dot{\boldsymbol{m}}^{\boldsymbol{t}})_{\boldsymbol{i}})$
21: Compute a_i^t with Equation (2)
22: **end while**

whether to be sent in Step 7∼17 as Subsection 4.4. After that, agent i can receive a set of messages from system agents in Step 18. Agent i overwrites the local estimated message vector using the received message set and aggregates information from the overwritten message vector as operated in Subsection 4.3. Finally the action a_i^t is generated in Step 21.

5 EXPERIMENTS

We evaluate MBC on three mixed cooperative-competitive environments which are widely utilized tasks in previous the state-of-the-art work [4, 19, 23, 32, 42]. These environments provide individual rewards for agents to motivate them to complete tasks. The details of these environments are as follows.

PP-grid. PP-grid is a predator-prey environment with grid information as agents' observation. In this environment, the 5 predators' policies are to be learned to move onto the fixed location of a prey. Each predator only has a view of the neighboring cells around it. A predator is unaware of the location of prey unless the prey appears in its view. Each time step every predator will receive a -0.05 reward unless it moves to the prey position where it receives 0 reward.

CN-loc. CN-loc is a navigation environment with relative locations of landmarks and other agents as individual observations. 7 agents in this task learn to occupy 7 landmarks. Each agent obtains partial observation of the environment. The individual reward is based on the minimum of negative distances of all landmarks to the individual agent. Besides, agents will receive individual -0.5 reward if they collide with each other.

PP-loc. PP-loc is a predator-prey environment with relative locations of preys and other predators as individual observation. 7 predators in this task learn to pursuit 3 preys. Each agent obtains partial observation of the environment. The individual reward is based on the minimum of negative distances of all preys to the individual agent. Different from PP-grid, the preys in PP-loc can move to get away from the near predators.

These three environments cover three diverse communication learning tasks. In PP-grid, agents need to learn when to broadcast location information of the prey. Sharing intentions with other agents to choose the particular target without collision is specially required in CN-loc. And PP-loc requires agents learn how to cooperate in capturing moving preys through communication.

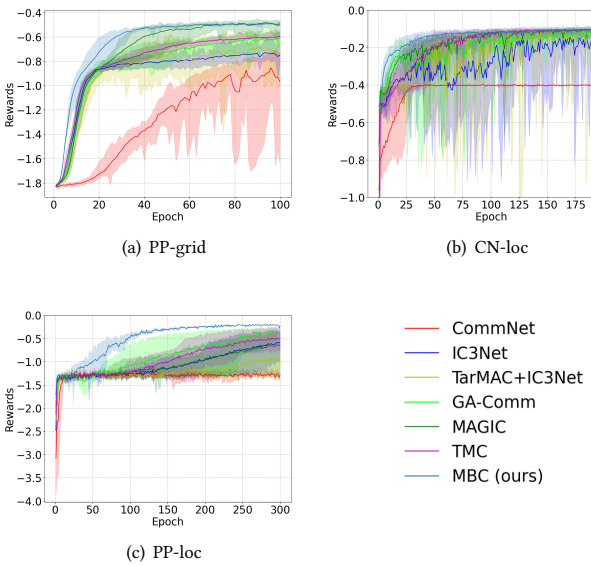
To demonstrate the superiority of MBC, we have chosen a variety of state-of-art communication MARL methods as our baselines. All of them target on solving mix cooperative-competitive tasks like ours. These include CommNet [33] that requires all agents to communicate with each other, IC3Net [32] for learning when to send messages, TarMAC+IC3Net [3] for learning aggregated messages using attention mechanism, GA-Comm [18] for learning communication graphs, MAGIC [23] for centralized message scheduling, and TMC [42] for decentralized message scheduling. Among these baselines, MAGIC is the state-of-the-art method in terms of agent performance and TMC is the state-of-the-art method in terms of efficient communication channel. Notably, we do not compare with the model-based MARL methods [30] [6], since our proposed method uses a model-free approach to learn the optimal policies and builds the message estimation model only for communication. Our method can be categorized as 'model-free learning with model-based communication'. To the best of our knowledge, our method is the first work in this category.

We use RMSProp with learning rate 0.00025 to train the Message Encoder, Decision Generator, and Message Aggregator jointly. We update 160 times for each time steps and each update is with 500 batch to perform batch learning. The size of sending message m_i and the aggregated messages am_i is 128. The neuron number in hidden layers of FC1, FC2 and LSTM is set to be 128. We use two layer GATs to implement the Message Aggregator. The first layer is consistent of 32 hidden neurons and 4 heads, while the second layer is consistent of 128 hidden neurons and 1 head. Besides, to train the Message Estimator, we use Adam with learning rate 0.001. The replay buffer $\mathcal{D}$ for Message Estimator is implemented by a queue with length 100K . The batch for sampling in Equation (5) is 128. The value coefficient β is set to 0.01, 0.015 and 0.01 in PP-grid, CN-loc and PP-loc, respectively. The discount γ is set to 1,

0.99, 0.99 in PP-grid, CN-loc and PP-loc, respectively. The error threshold of the scheduling message in Subsection 4.4 is set to be 0.05, 0.2 and 0.2 in PP-grid, CN-loc and PP-loc, respectively. The window size is set to 40, as same as the maximum time step of an episode. All evaluation results of of each algorithm are come from 10 independent experiments initialized with different random seeds from 2022 ~ 2031. Our code is open source[1].

5.1 Performance

We first compare the performance of MBC with the baselines to validate the superiority of MBC in solving MARL tasks. The performance of MBC and all baselines are presented in Figure 3. The error



(a) PP-grid (b) CN-loc

(c) PP-loc

Figure 3: Learning curves of various communication MARL algorithms in three mixed cooperative-competitive environments.

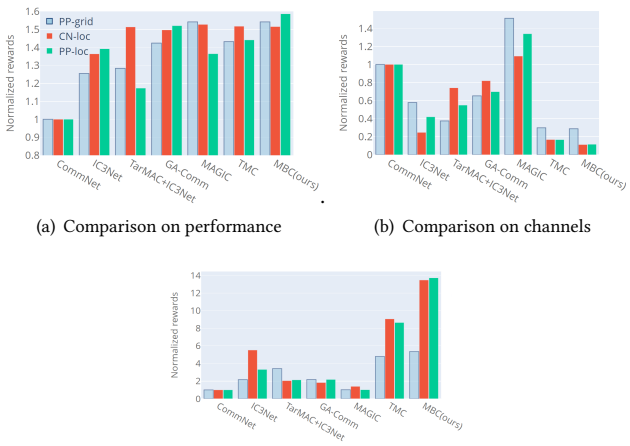
bounds (i.e., shadow shapes) indicate the upper and lower bounds of the performance with 10 runs. In PP-grid and CN-loc, the prey and landmarks are stationary. In these environments, our approach achieves similar performance to the previous baselines, while our approach shows better sample efficiency. When it comes to the more challenging PP-loc environment in which preys are moving, MBC significantly outperforms previous methods and exhibits a smaller performance variance.

5.2 Communication Efficiency

We evaluate the communication efficiency of MBC in this subsection. We define the communication efficiency using the ratio of performance to channel overhead. The larger this ratio is, the better ability to achieve higher performance using fewer channels for the algorithm.

Figure 4 compares the communication efficiency of the algorithms. To avoid calculating negative rewards, we take the al-

[1]https://github.com/shan0126/Model-Based-Communication



(a) Comparison on performance (b) Comparison on channels

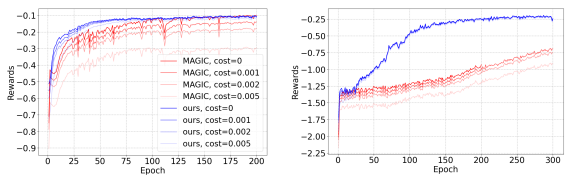
(c) Comparison on ratio of performance to channels

Figure 4: Communication efficiency comparison of various communication MARL algorithms in three mixed cooperative-competitive environments.

gorithms' reward increment over random policy and normalize them over CommNet in the way similar to [35]: $r_{normalized}(alg) = \frac{r_{alg}-r_{random}}{r_{CommNet}-r_{random}}$, where $r_{normalized}(alg)$ is normalized reward for any alg method to be compared, r_{alg} is the average reward achieved by alg method, $r_{CommNet}$ and r_{random} are the average rewards achieved by CommNet method and by a random policy, respectively. In our experiments, we consider a message from one agent to another as one communication channel. The normalized channel $c_{normalized}(alg)$ for method alg is calculated by: $c_{normalized}(alg) = \frac{c_{alg}}{N(N-1)}$, where c_{alg} is the average channel consumed by alg method and N is the number of agent in the system.

Figure 4(a) and 4(b) shows the performance and communication channel of each algorithm. Figure 4(c) presents the ratio of performance to channel overhead. The results show our MBC method can achieve higher performance with less channel overhead. Figure 4(c) demonstrates the significant improvement in communication efficiency of MBC compared to baselines.

As motivated in the introduction, achieving good performance with fewer channels is especially important in environments where communications are costly. Figure 5 shows the learning curves of MBC and MAGIC for a verity of environments where the cost per communication channel is 0.001, 0.002 and 0.005 respectively. When



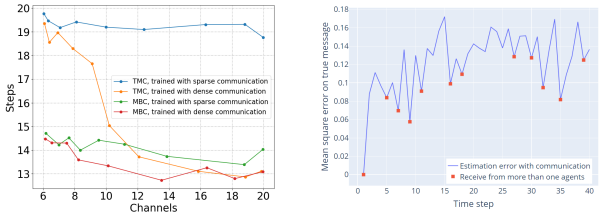
(a) CN-loc (b) PP-loc

Figure 5: The learning curves with communication overhead is counted in the reward.

the cost of the channel increases, the learning curve of MAGIC shifts significantly downward, whereas the learning curve of MBC only becomes slightly lower. This indicates that our approach is more stable and more efficient in environments where communication is costly. It further validate the feasibility of our method.

5.3 Channel Dependency

In real-world situations, agents may also need to accomplish tasks with restricted communication channels. Thus, we further validate the feasibility of our method by examining the influence of communication channel restriction on the performance of MBC. We compare our MBC with TMC that has the lowest communication overhead baseline in terms of channel dependency. The performance changes of two methods over different channels are shown in Figure 6(a). We limit the allowed communication channels in the training. Dense communication basically allows full communication, while sparse only allows up to 6 channels. We use these learned models to perform the task and record how many steps are needed to complete the task when we allow different numbers of communication channels. The horizontal axis of Figure 6(a) is average channel overhead per time step, and the vertical axis is the steps required for all predators to catch the prey in the PP-grid environment. The smaller the required steps, the more efficient the algorithm is in completing the task.



(a) The achieved performance changes over channels in PP-grid.

(b) Estimation error of agent with communication.

Figure 6: Performance changes over channels and estimation error changes over time steps.

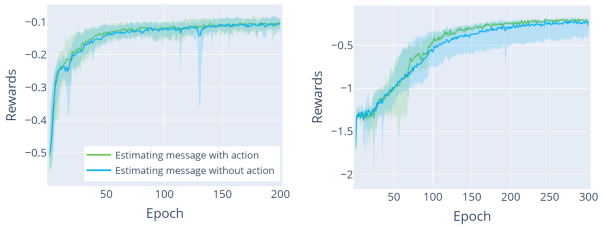
As shown in Figure 7, with sparse communication training, TMC always requires more steps to complete the task than MBC in execution (compare blue and green lines). With dense communication training, TMC and MBC can achieve similar performance in the execution when there are enough communication channels. However, when there are fewer communication channels available, the efficiency of the TMC to complete the task becomes significantly lower, while the performance of MBC is only slightly reduced, as shown on the orange and red lines. In summary, MBC is more capable of communication with constraints on channels.

5.4 Interpretation on Message Scheduling

A remarkable feature of MBC is that communication is used to reduce estimation error on global message. Thus, we investigate the effects of communication (i.e. received messages) on the message estimation error. To do this, we track dynamic changes of the estimation error on global messages in one agent and plot them in Figure 6(b). As shown in the figure, the estimation error is decreased when more then one messages are received from other agents. This indicate the effectiveness of communication on correcting local estimation on global message.

5.5 Ablation Experiment

To explore the effect of joint action on model estimation in Equation (6), we compare the method estimating messages using Equation (6) and that estimating messages using $(\hat{m}^t)_i = f_M\left((\dot{m}^{t-1})_i\right)$ without actions. Overall, as shown in Figure 7 the method estimating message with actions shows better performance. This advantage



(a) CN-loc

(b) PP-loc

Figure 7: The learning curves of MBC with and without action for estimating message.

is not obvious in CN-loc, because landmarks in the environment are stationary and their position is not affected by agents' actions. However, when it comes to moving prey based on agents actions, estimating information without action makes the method more unstable. This means that the usage of action to estimate message changes is more essential.

6 CONCLUSION AND FUTURE WORK

In this paper, we propose MBC, a decentralized communication framework in MARL with sparse communication. MBC utilizes a message model to estimate up-to-date messages of other agents instead of always receiving messages from them. A decentralized message scheduling mechanism is also designed to correct the error of the agent's message estimation. The proposed method allows multiple agents to make collaborative decisions with sparse communication. In a variety of mixed cooperative-competitive environments, MBC shows around 5.16% better performance and around 22.33% lower communication overhead than the state-of-the-art method. We want to continue our efforts to decrease the communication overhead of MARL methods. In this paper, we only use the message model to estimate the observations and intentions of other agents at this moment. In future work, we would like to extend this model to predict the future observations and intentions of agents for assisting their local decision making.

ACKNOWLEDGEMENTS

We sincerely thank the anonymous reviewers. This work is partly funded by the China Scholarship Council (CSC).

REFERENCES

[1] Akshat Agarwal, Sumit Kumar, Katia P. Sycara, and Michael Lewis. 2020. Learning Transferable Cooperative Behavior in Multi-Agent Teams. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1741–1743.

[2] Petar Veličković Guillem Cucurull Arantxa Casanova, Adriana Romero Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *6th International Conference on Learning Representations* (2018).

[3] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. 2019. TarMAC: Targeted Multi-Agent Communication. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. 1538–1546.

[4] Ziluo Ding, Tiejun Huang, and Zongqing Lu. 2020. Learning Individually Inferred Communication for Multi-Agent Cooperation. In *Advances in Neural Information Processing Systems*.

[5] Yali Du, Bo Liu, Vincent Moens, Ziqi Liu, Zhicheng Ren, Jun Wang, Xu Chen, and Haifeng Zhang. 2021. Learning Correlated Communication Topology in Multi-Agent Reinforcement learning. In *20th International Conference on Autonomous Agents and Multiagent Systems*. 456–464.

[6] Vladimir Egorov and Alexey Shpilman. 2022. Scalable Multi-Agent Model-Based Reinforcement Learning. In *21st International Conference on Autonomous Agents and Multiagent Systems*. 381–390.

[7] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2018. Counterfactual Multi-Agent Policy Gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. 2974–2982.

[8] Sven Gronauer and Klaus Diepold. 2022. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review* 55, 2 (2022), 895–943.

[9] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation*. 3389–3396.

[10] Shushi Gu, Ye Wang, Niannian Wang, and Wen Wu. 2020. Intelligent optimization of availability and communication cost in satellite-UAV mobile edge caching system with fault-tolerant codes. *IEEE Transactions on Cognitive Communications and Networking* 6, 4 (2020), 1230–1241.

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[12] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. 2019. When to Trust Your Model: Model-Based Policy Optimization. In *Advances in Neural Information Processing Systems 32*. 12498–12509.

[13] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Çaglar Gülçehre, Pedro A. Ortega, DJ Strouse, Joel Z. Leibo, and Nando de Freitas. 2019. Social Influence as Intrinsic Motivation for Multi-Agent Deep Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. 3040–3049.

[14] Jiechuan Jiang and Zongqing Lu. 2018. Learning Attentional Communication for Multi-Agent Cooperation. In *Advances in Neural Information Processing Systems*. 7265–7275.

[15] Daewoo Kim, Sangwoo Moon, David Hostallero, Wan Ju Kang, Taeyoung Lee, Kyunghwan Son, and Yung Yi. 2019. Learning to Schedule Communication in Multi-agent Reinforcement Learning. In *7th International Conference on Learning Representations*.

[16] Woojun Kim, Jongeui Park, and Youngchul Sung. 2021. Communication in Multi-Agent Reinforcement Learning: Intention Sharing. In *9th International Conference on Learning Representations*.

[17] Michael L Littman. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings*. 157–163.

[18] Yong Liu, Weixun Wang, Yujing Hu, Jianye Hao, Xingguo Chen, and Yang Gao. 2020. Multi-Agent Game Abstraction via Graph Attention Neural Network. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*. 7211–7218.

[19] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems*. 6379–6390.

[20] Hangyu Mao, Zhengchao Zhang, Zhen Xiao, Zhibo Gong, and Yan Ni. 2020. Learning Agent Communication under Limited Bandwidth by Message Pruning. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*. 5142–5149.

[21] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33nd International Conference on Machine Learning*, Vol. 48. 1928–1937.

[22] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. 2018. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. In *2018 IEEE International Conference on Robotics and Automation*. 7559–7566.

[23] Yaru Niu, Rohan Paleja, and Matthew Gombolay. 2021. Multi-Agent Graph-Attention Communication and Teaming. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. 964–973.

[24] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. 2018. Activation Functions: Comparison of trends in Practice and Research for Deep Learning. *CoRR* abs/1811.03378 (2018).

[25] Georgios Papoudakis, Filippos Christianos, and Stefano V. Albrecht. 2021. Agent Modelling under Partial Observability for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems 34*. 19210–19222.

[26] Bei Peng, Tabish Rashid, Christian Schröder de Witt, Pierre-Alexandre Kamienny, Philip H. S. Torr, Wendelin Boehmer, and Shimon Whiteson. 2021. FACMAC: Factored Multi-Agent Centralised Policy Gradients. In *Advances in Neural Information Processing Systems*. 12208–12221.

[27] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. 2018. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80. 4292–4301.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017).

[29] Esmaeil Seraj, Zheyuan Wang, Rohan R. Paleja, Daniel Martin, Matthew Sklar, Anirudh Patel, and Matthew C. Gombolay. 2022. Learning Efficient Diverse Communication for Cooperative Heterogeneous Teaming. In *21st International Conference on Autonomous Agents and Multiagent Systems*. 1173–1182.

[30] Pier Giuseppe Sessa, Maryam Kamgarpour, and Andreas Krause. 2022. Efficient Model-based Multi-agent Reinforcement Learning via Optimistic Equilibrium Computation. In *International Conference on Machine Learning*, Vol. 162. 19580–19597.

[31] Piyush K. Sharma, Rolando Fernandez, Erin G. Zaroukian, Michael R. Dorothy, Anjon Basak, and Derrik E. Asher. 2021. Survey of Recent Multi-Agent Reinforcement Learning Algorithms Utilizing Centralized Training. *CoRR* abs/2107.14316 (2021).

[32] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. 2019. Learning when to Communicate at Scale in Multiagent Cooperative and Competitive Tasks. In *7th International Conference on Learning Representations*.

[33] Sainbayar Sukhbaatar, Rob Fergus, et al. 2016. Learning multiagent communication with backpropagation. *Advances in Neural Information Processing Systems* 29, 2244–2252.

[34] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinícius Flores Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. 2018. Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. 2085–2087.

[35] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. 2094–2100.

[36] Akifumi Wachi. 2019. Failure-Scenario Maker for Rule-Based Agent using Multi-agent Adversarial Reinforcement Learning and its Application to Autonomous Driving. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. 6006–6012.

[37] Zixin Wang, Hanyu Zhu, Mingcheng He, Yong Zhou, Xiliang Luo, and Ning Zhang. 2022. GAN and Multi-Agent DRL Based Decentralized Traffic Light Signal Control. *IEEE Transactions on Vehicular Technology* 71, 2 (2022), 1333–1348.

[38] Daniël Willemsen, Mario Coppola, and Guido C. H. E. de Croon. 2021. MAMBPO: Sample-efficient multi-robot reinforcement learning using learned world models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 5635–5640.

[39] Chongjie Zhang and Victor R. Lesser. 2013. Coordinating multi-agent reinforcement learning with limited communication. In *International conference on Autonomous Agents and Multi-Agent Systems*. 1101–1108.

[40] Sai Qian Zhang, Qi Zhang, and Jieyu Lin. 2019. Efficient Communication in Multi-Agent Reinforcement Learning via Variance Based Control. In *Advances in Neural Information Processing Systems 32*. 3230–3239.

[41] Sai Qian Zhang, Qi Zhang, and Jieyu Lin. 2019. Efficient Communication in Multi-Agent Reinforcement Learning via Variance Based Control. In *Advances in Neural Information Processing Systems*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 3230–3239.

[42] Sai Qian Zhang, Qi Zhang, and Jieyu Lin. 2020. Succinct and Robust Multi-Agent Communication With Temporal Message Control. In *Advances in Neural Information Processing Systems*, Vol. 33. 17271–17282.

[43] Weijia Zhang, Hao Liu, Jindong Han, Yong Ge, and Hui Xiong. 2022. Multi-Agent Graph Convolutional Reinforcement Learning for Dynamic Electric Vehicle Charging Pricing. In *The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2471–2481.

[44] Changxi Zhu, Mehdi Dastani, and Shihan Wang. 2022. A Survey of Multi-Agent Reinforcement Learning with Communication. *CoRR* abs/2203.08975 (2022).