B.Eng. Dissertation

Multi-agent strategy learning in reinforcement learning for coordinated problem solving

Submitted by

Wira Azmoon Ahmad

In partial fulfilment of the

requirements for the Degree of

Bachelor of Engineering (Computer Engineering)

School of Computing

National University of Singapore

2022/2023

B.Eng. Dissertation

Multi-agent strategy learning in reinforcement learning for coordinated problem solving

By

Wira Azmoon Ahmad

School of Computing

National University of Singapore

2022/2023

Project ID: H247110

Project Supervisor: Dr Akshay Narayan

Deliverables:

Report: 1 Volume

Abstract

This work aims to investigate the use of Multi-Agent Reinforcement Learning (MARL) for coordinated problem solving by exploring the effectiveness of various algorithms. MARL is a subfield of Machine Learning where multiple agents learn to solve a problem together, that involves using RL concepts (Sutton & Barto, 2018), such as the Markov Decision Process (MDP), value functions, reward functions, and learning from environments. The work will focus on developing and evaluating different MARL algorithms that communicate (Comm-MARL) with the aim of knowledge sharing and coordination in various environments, implementing those MARL algorithms, and testing their performance on a common benchmark. The findings will contribute to the benchmarking of a Comm-MARL algorithm in an established MARL framework, and can provide a stepping stone to further development.

Subject Descriptors:

(1) Theory of computation~Theory and algorithms for application domains~Machine learning theory~Reinforcement learning~Multi-agent reinforcement learning;500

(2) Computing methodologies~Machine learning~Machine learning approaches~Neural networks;500

(3) Theory of computation~Theory and algorithms for application domains~Machine learning theory~Markov decision processes;300

Keywords:

Multi-agent, reinforcement learning, neural networks, Markov decision process, machine learning

Implementation Software and Hardware:

Linux Ubuntu 20.04.6 LTS, Python 3.8.10, PyTorch 1.13, OpenAI Gym 0.21.0, CUDA 11.7, NVIDIA A100 GPU

Table of Contents

1 Introduction

Reinforcement Learning (RL), under the umbrella of Machine Learning paradigms, is at its core about learning what actions to take, under a certain situation (Sutton & Barto, 2018). An agent would attempt to learn about its environment over time, using a reward system to achieve a final goal. The reward system provides rewards to measure the goodness of taking actions while being in a particular state within the environment. Multi-Agent RL (MARL) involves multiple agents doing such learning together, where these agents could either be cooperative or competitive, working toward achieving the goal of the system. With the rise of Deep Learning, Deep MARL has also contributed to the popularity of the field, where performance in existing environments are improved even further.

Oroojlooy & Hajinezhad (2019) reviewed cooperative MARL, defining a taxonomy for the sub-field and detailing the various approaches to developing algorithms that solve related problems. Zhu, Dastani, & Wang (2022) reviewed MARL with communication (Comm-MARL), considering various aspects of communication which can affect the various MARL algorithms and their designs or development. This project aims to explore the algorithms used in cooperative MARL, apply them to environments related to the context of coordination and communication among agents to solve problems, fairly compare the various algorithms on a common baseline benchmark, and identify issues faced in the chosen algorithms to further improve on them.

1.1 Motivation

Cooperative MARL has applications in controlling traffic, resource allocation, robot path planning, production systems, image classification, the stock market, and maintenenace management (Oroojlooy & Hajinezhad, 2019). A hive of search and rescue (SAR) robots can demonstrate the complexity in the simple task of rescuing a human being. In this simple example, multiple robots need to execute their own actions to achieve the goal, taking into account the dynamic environment with different possibilities of the physical area, the weight of the human, and the actions of the other robots in the hive, among many possibilities. Power generators can also demonstrate MARL's use in terms of resource allocation. The group of power generators must work concurrently to efficiently produce a certain minimum threshold of power, without wasting energy. In these examples, there is the common factor of modeling the environment as having multiple agents which are required to execute their own actions to achieve a common goal. The next section details further on how these environments could be modeled, using notation used in RL.

2 Background

Before exploring the methods used to research RL, the background and notation of RL will be detailed in the next few sections.

2.1 Single-Agent Setting

The single-agent setting can be represented by the tuple $\langle \mathcal{S}, \mathcal{A}, R, P, \gamma \rangle$, with the mathematical framework of the Markov Decision Process (MDP). An agent interacting with an environment will, at each time step t, observe a state $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the state space, performs an action $a_t \in \mathcal{A}(s_t)$ where $\mathcal{A}(s_t)$ is the valid action space for state s_t, and receive a reward $R(s_t, a_t, s_{t+1}) \in \mathbb{R}$ before transitioning to the new state $s_{t+1} \in \mathcal{S}$. There is an associated transition model $P(s_t, a_t, s_{t+1}) \sim \Pr(s_{t+1}|s_t, a_t)$ which gives the probability to go from state s_t to state s_{t+1}, when taking the action a_t. The underlying assumption used in MDP is the Markov assumption, where the next state is solely based on the current state and action. The goal is to determine a policy π for the agent, which can be deterministic, where $\pi(s_t) = a_t$, or stochastic, where $\pi(s_t) \sim \Pr(a|s_t)$, the probability distribution for actions given the state s_t.

Each state $s \in \mathcal{S}$ has a value for a policy π defined by the value function $V^\pi\colon \mathcal{S} \to \mathbb{R}$

$$V^\pi(s) = \mathbb{E}_\pi[R_t|\, a_t \sim \pi(s_t), s_0 = s] \tag{1}$$

where $R_t = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$ is the discounted return for a discount factor $\gamma \in [0,1)$.

Accordingly, the Q-value function $Q^\pi\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for each state-action pair is defined as

$$Q^\pi(s,a) = \mathbb{E}_\pi[R_t|\, a_t \sim \pi(s_t), s_0 = s, a_0 = a] \tag{2}$$

A standard definition for the advantage function $A^\pi\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ can then be (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017)

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{3}$$

With the optimal policy π^*, the optimal value and Q-value functions can be obtained

$$V^{\pi^*}(s) = \max_a \sum_{s'} \Pr(s'|a,s) \left[R(s,a,s') + \gamma V^{\pi^*}(s')\right] \tag{4}$$

$$Q^{\pi^*}(s,a) = \sum_{s'} \Pr(s'|a,s) \left[R(s,a,s') + \gamma \max_{a'} Q^{\pi^*}(s',a')\right] \tag{5}$$

The functions are then related via

$$V^{\pi^*}(s) = \max_a Q^{\pi^*}(s,a) \tag{6}$$

with the optimal policy

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^{\pi^*}(s,a) \tag{7}$$

2.2 Multi-Agent Setting

In MARL, although simpler problems can be modelled to have fully observable environments for all agents, it is more common for agents to not be able to observe the full environment state. Thus, most problems in the multi-agent setting can be described as a Decentralized Partially Observable MDP (Dec-POMDP) (Ong, Png, Hsu, & Lee, 2009) with the tuple $(n, \mathcal{S}, \mathcal{A}, R, P, \mathcal{O}, \gamma)$, where n is the number of agents, $\mathcal{S}$ is the state space, $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \ldots \times \mathcal{A}_n$ is the set of actions for all agents, R is the reward function, P is the transition probability among states, $\mathcal{O} = \mathcal{O}_1 \times \mathcal{O}_2 \times \ldots \times \mathcal{O}_n$ is the set of observations for all agents, and γ is the discount factor. We denote $a_i \in \mathcal{A}_i$ to be an action for agent i, while $\boldsymbol{a} \in \mathcal{A}$ is the combined action vector. Similarly, $o_i \in \mathcal{O}_i$ and $\boldsymbol{o} \in \mathcal{O}$. Additionally, $\mathcal{T} \equiv (\mathcal{O} \times \mathcal{A})^*$ is the observation-action space, where $\tau_i \in \mathcal{T}_i$ is the observation-action history for agent i, and $\boldsymbol{\tau} \in \mathcal{T}$ is a combined observation-action history. Each agent can observe its own local reward $R_i\colon \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \rightarrow \mathbb{R}$ and learn a policy $\pi_i(a_i|o_i) \sim \Pr(a_i|o_i)$.

2.3 Value Approximation

One way to approach the single-agent RL problem is to first use Value Approximation, which aims to model $V(s)$ or $Q(s,a)$ by approximating the functions. While simpler function approximators such as linear approximators could be used (Sutton & Barto, 2018), recent development in the state-of-the-art involves deep learning. In Deep RL, a neural network can be used to model $Q(s,a)$ like in Deep Q-Networks (DQN) (Mnih, et al., 2015). An experience replay buffer $\mathcal{D}$ will hold tuples $(s,a,r,s') \in \mathcal{D}$, and a loss function which will be minimized for some parameters θ can be defined as follows.

$$L(\theta) = \mathbb{E}_{s,a\sim\mathcal{D}}\left[\left(y - Q(s,a;\theta)\right)^2\right] \tag{8}$$

$$y = r + \gamma \max_{a'} Q(s',a';\theta^-) \tag{9}$$

θ^- represents the parameters of the target network, updated by θ every C iterations. The replay buffer and the use of θ^- help to stabilize learning. Here, DQN is an off-policy algorithm.

In the multi-agent setting, the replay buffer $\mathcal{D}$ can store larger tuples $(\boldsymbol{o},\boldsymbol{a},\boldsymbol{r},\boldsymbol{o}')$ to keep the different actions and rewards for the various agents, each with their own local observations. The loss function simply considers each agent's Q-value function Q_i and their own loss function for parameters $\theta = (\theta_1, \dots, \theta_n)$. In the Multi-Agent Deterministic Policy Gradient (MADDPG) algorithm, the loss function is defined as follows (Lowe, et al., 2017), like the DQN loss function above.

$$L(\theta_i) = \mathbb{E}_{\boldsymbol{o},\boldsymbol{a},\boldsymbol{r},\boldsymbol{o}'}\left[\left(y - Q_i(\boldsymbol{o},\boldsymbol{a};\theta_i)\right)^2\right] \tag{10}$$

$$y = r_i + \gamma Q_i(\boldsymbol{o}',\boldsymbol{a}';\theta_i')\big|_{a_j'=\pi_j'(o_j')} \tag{11}$$

$\pi' = \{\pi_1', \dots, \pi_n'\}$ is the set of target policies that are making use of the delayed parameters θ_i'.

2.4 Policy Approximation

Instead of estimating the value functions, Policy Approximation can directly parameterize the policy and optimize a utility function over the policy parameter. The function $J(\theta)$ can be defined as the expected value of policy π_θ for a trajectory of T timesteps $\tau = s_0, a_0, r_0, \dots, s_T, a_T, r_T$.

$$J(\theta) = \mathbb{E}_\tau[R] \tag{12}$$

Steps are taken in the direction of $\nabla J(\theta)$ to improve optimize the value of $J(\theta)$, which can be then defined as (Sutton, McAllester, Singh, & Mansour, 1999)

$$\nabla J(\theta) = \mathbb{E}_{s,a\sim\mathcal{D}}[\nabla_\theta \log \pi_\theta(a|s)\, Q^\pi(s,a)]. \tag{13}$$

For the multi-agent setting, the gradient for expected return for agent i can be written to include multiple agents. As an example, in MADDPG, it is written as follows (Lowe, et al., 2017).

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{o},\boldsymbol{a}\sim\mathcal{D}}\left[\nabla_{\theta_i}\pi_i(a_i|o_i)\nabla_{a_i} Q_i^\pi(\boldsymbol{o},\boldsymbol{a}) \mid a_i = \pi_i(o_i)\right] \tag{14}$$

The key difference between approximating the Q-value function in Policy and Value Approximation is that Value Approximation aims to find the true optimal Q^*-value function, while this may not be the case in Policy Approximation. For example, if Policy Approximation uses an estimator $\hat{Q} = Q^*/100$, the function would work as well as the optimal Q^* function in obtaining a good policy, even though the absolute difference in values would be significant.

2.5 DRQN

With the rise of powerful computing and capable hardware to support the effectiveness of Deep Learning, Deep Recurrent Q-Networks (DRQN) are a natural progression to the simpler DQNs employed by the earlier MARL algorithms (Hausknecht & Stone, 2015). Typical DQNs can approximate $Q(s,a)$, but partial observability can cause the assumed state of the environment used as input to be inaccurate. By leveraging the benefits of a recurrent neural network (RNN), a DRQN can instead be used to approximate $Q(o_t, h_{t-1}, a)$, maintaining its own internal state and aggregate observations over time. DRQNs themselves make use of Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) with the DQN, addressing the partial observability of POMDPs.

3 Literature Review

MARL algorithms can be split into five categories – Independent Learners, Fully Observable Critic, Value Function Factorization, Consensus, and Learn to Communicate (Oroojlooy & Hajinezhad, 2019).

Independent Learners treat each agent independently and other agents' actions to be a part of the environment. A large problem with this approach is the problem of non-stationarity of the environment, since other agents' actions will impact the changes in the environment. Non-stationarity means that $\Pr(s'|s, a_i, \pi_1, \dots, \pi_n) \neq \Pr(s'|s, a_i, \pi'_1, \dots, \pi'_n)$ if any $\pi_i \neq \pi'_i$, violating the Markov assumption for each agent i. The DQN algorithm can be used in Independent Q-Learning (IQL) to achieve superhuman-level control in Atari games (Mnih, et al., 2015).

With a Fully Observable Critic, the non-stationarity issue is addressed since all observations and actions of all agents are seen by the critic. This is because the next state is independent of the policy, given the agents' actions, where $\Pr(s'|s, a_1, \dots, a_n, \pi_1, \dots, \pi_n) = \Pr(s'|s, a_1, \dots, a_n)$. The critic can ultimately act as a leader for the local actors of the model. Depending on the reward system, either 1 or n critics can be trained. A different local reward for each agent may require n critics for each agent, while fully cooperative agents would make use of a single critic instead. The algorithms in this category are also often referred to as actor-critic models.

In problems that may have a single joint reward or that can be reduced to a single agent problem with a combined action space, an RL algorithm may face a problem of having lazy agents which do not learn (Sunehag, et al., 2018). Value Function Factorization aims to solve this by determining the share of the value function that each agent contributes.

Consensus and Learn to Communicate algorithms factor in the communication between agents when solving the MARL problems, which can be useful in situations with communication cost and bandwidth limitations.

Learn to Communicate algorithms typically make use of the concept of message (or communication-action) passing between agents to communicate – referred to as MARL with communication (Comm-MARL). The agents themselves would learn to send the messages, which contain knowledge or information, that they deem relevant, based on the architecture given by the algorithm. Typically, the messages are determined by each agent after receiving the observation and passed on to other agents directly or indirectly. For example, the architecture can make use of gradient flow (Foerster, Assael, de Freitas, & Whiteson, 2016) or a common knowledge source (Liu, et al., 2023) to communicate messages from agent to agent. Although messages usually come from each agent with the output actions, the messages would not affect the next state or reward of each agent. The environment and agent interaction would be the same, with the added communication between agents improving performance by taking more informed actions.

In this project, we study the Independent Learner and Fully Observable Critic categories to understand the state-of-the-art algorithms and baseline performance more deeply in the more ideal setting where communication cost is not a limiting factor, followed by the Value Function Factorization category which aims to address the specific problem of lazy agents. Learn to Communicate algorithms are then studied as these algorithms will allow us to better improve a mechanism for knowledge transfer and coordination between agents. Algorithms in this last category generally build on the baseline algorithms and performance provided by the previous categories, including specific additions that can facilitate the flow of information. Based on the current literature and results found in benchmarks (Papoudakis, Christianos, Schäfer, & Albrecht, 2021), Fully Observable Critics provide a solid basis of comparison for future exploration on algorithms that better address the problem of non-stationarity. The systematic and structural methodologies for Comm-MARL algorithms are only recently being established (Zhu, Dastani, & Wang, 2022), implying the availability of further research and development in the sub-field to introduce benchmarks, frameworks, and baselines. A Comm-MARL algorithm is then chosen and compared with the baseline, with the goal of being improved upon, leveraging the benefits of communication as mentioned earlier.

4 Methods

Following from the general taxonomy of algorithms used in MARL, this chapter details the mechanisms of various chosen algorithms, along with the environments used in the state-of-the-art benchmarks. Algorithms in RL are commonly evaluated using simulated environments, along with difficult repeated matrix games (Claus & Boutilier, 1998).

4.1 Algorithms

The algorithms discussed make use of the paradigm of Centralized Training with Decentralized Execution (CTDE), where the agents are trained using centralized information, but the execution of actions is decentralized. This has achieved widespread adoption and popularity with actor-critic models (Lyu, Xiao, Daley, & Amato, 2021), but are also used in the Value Function Factorization algorithms.

4.1.1 MAA2C

The Independent Synchronous Advantage Actor-Critic (IA2C) is a variant of the original Asynchronous Advantage Actor-Critic (A3C) (Mnih, et al., 2016). A3C has an actor $\pi_i(a_t|s_t;\theta)$ and a critic which estimates the value function $V(s_t;\theta_v)$. The gradient of the objective function used is

$$\nabla J(\theta) = \mathbb{E}_{s,a\sim\mathcal{D}}[\nabla_{\theta'} \log \pi(a|s;\theta')\, A(s,a;\theta,\theta_v)], \tag{15}$$

using an estimated advantage function $A(s_t,a_t;\theta,\theta_v) = R_t - V(s_t;\theta_v)$ given by

$$A(s_t,a_t;\theta,\theta_v) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v). \tag{16}$$

k can vary from state to state and is upper bounded by a hyperparameter t_{max}. Both $\pi(a_t|s_t;\theta)$ and $V(s_t;\theta_v)$ are estimated by neural networks with some shared parameters, although they may be chosen to not be shared in the general case.

Adding the entropy of the policy π can also improve exploration, so the full objective function with the entropy regularization term added with H as the entropy is then

$$\nabla J(\theta) = \mathbb{E}_{s,a\sim\mathcal{D}}\left[\nabla_{\theta'} \log \pi_i(a|s;\theta')\, A(s,a;\theta,\theta_v) + \beta \nabla_{\theta'} H\big(\pi(s_t;\theta')\big)\right]. \tag{17}$$

β is a hyperparameter controlling the strength of regularization.

While the original paper (Mnih, et al., 2016) used asynchrony to implement A3C, it was found that this does not provide much advantage over a synchronous and deterministic implementation (Wu, Mansimov, Liao, Grosse, & Ba, 2017). Thus, A2C is used as the baseline algorithm which can be built upon for the multi-agent setting.

Independent learning with multiple agents of IA2C means that each agent i has its own actor $\pi_i(a_i|s_i;\theta_i)$ and critic networks $V_i(s_i;\theta_{vi})$ to approximate the policy and value functions respectively, where θ_i denotes parameters for the policy π_i, and θ_{vi} denotes parameters for the value function V_i.

Multi-Agent A2C (MAA2C) is a simple extension of IA2C which uses a centralized joint state-value function $V(\boldsymbol{s};\theta_v)$ as its critic. The critic is conditioned on the environment states, and does not make use of each agent's actions, unlike MADDPG as seen in the next section.

4.1.2 MADDPG

MADDPG is an actor-critic algorithm in a setting where each agent i at each time step has its own local observation o_i, action a_i, and reward a_i. Each agent has an actor $\pi_i(o_i;\theta_i)$ that executes an action for each observation, while a centralized critic Q_i^π can access the observations, actions, and target policies of all agents when training. The Deterministic Policy Gradient (DPG) (Silver, et al., 2014) algorithm writes the gradient of the objective $J(\theta)$ as

$$\nabla J(\theta) = \mathbb{E}_{s,a\sim\mathcal{D}}[\nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s,a) \mid a = \pi_\theta(s)]. \tag{18}$$

Deep DPG (DDPG) approximates the actor π and critic Q^π using neural networks. Multi-Agent DDPG (MADDPG) would then extend the gradient for each agent to (Lowe, et al., 2017)

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{o},\boldsymbol{a}\sim\mathcal{D}}\left[\nabla_{\theta_i}\pi_i(a_i|o_i)\nabla_{a_i}Q_i^\pi(\boldsymbol{o},\boldsymbol{a}) \mid a_i = \pi_i(o_i)\right]. \tag{19}$$

The loss function is as previously detailed in Equations (10), (11),

$$L(\theta_i) = \mathbb{E}_{\boldsymbol{o},\boldsymbol{a},r,\boldsymbol{o}'}\left[\left(y - Q_i(\boldsymbol{o},\boldsymbol{a};\theta_i)\right)^2\right]$$

$$y = r_i + \gamma Q_i(\boldsymbol{o}',\boldsymbol{a}';\theta_i')|_{a_j'=\pi_j'(o_j')}$$

This assumes that each agent would know other agents' policies, since the combined action vector of all agents $\boldsymbol{a}$ is required. Another approach is that each agent i can maintain its own approximation of the policy of each agent j, $\hat{\pi}_i^j$, using parameters ϕ_i^j, to the true policy π_j. The approximate policy is optimized by maximizing the log probability of agent j's actions, with the entropy H of the policy distribution.

$$L(\phi_i^j) = -\mathbb{E}_{o_j,a_j}\left[\log \hat{\pi}_i^j\left(a_j \middle| oj\right) + \lambda H\left(\hat{\pi}_i^j\right)\right] \tag{20}$$

The loss function can instead use the approximate policies.

$$y = r_i + \gamma Q_i(\boldsymbol{o}', \hat{\pi}_i'^1(o_1), \dots, \pi'(o_i), \dots, \hat{\pi}_i'^n(o_n)) \tag{21}$$

This function can be optimized online, instead of requiring offline capture into the replay buffer, since the latest samples for each agent can be obtained to update ϕ_i^j before updating Q_i^π.

4.1.3 MAPPO

Multi-Agent Proximal Policy Optimization (MAPPO) (Yu, et al., 2021) is an actor-critic algorithm which extends the Independent Proximal Policy Optimization (IPPO) algorithm. PPO seeks to improve on Trust Region Policy Optimization (TRPO) (Schulman, Levine, Moritz, Jordan, & Abbeel, 2015), itself a policy gradient method that aims to maximize an object function subject to a constraint on the size of the policy update (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). PPO is architecturally identical to A2C, except that PPO constrains the relative change of the policy at each update. PPO maximizes the following objective function over the parameter θ with policy π_θ,

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)], \tag{22}$$

where L_t^{CLIP} is the surrogate clip objective using the advantage function $\hat{A}_t$ and a hyperparameter ϵ

$$L_t^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right], \tag{23}$$

L_t^{VF} is the squared error loss

$$L_t^{VF}(\theta) = \left(V_\theta(s_t) - V_t^{targ}\right)^2, \tag{24}$$

c_1, c_2 are coefficients, and S is an entropy bonus.

PPO can use the same batch of trajectories to update more epochs due to the constraint, differentiating it from A2C which does a single epoch update per batch of trajectory to remain on-policy.

MAPPO then learns the policy $\pi_i(a_i|o_i;\theta_i)$, for every agent i to execute action a_i at an observation o_i, with a joint state-value function $V_{\theta_v}(s)$, extending IPPO by considering multiple agents.

4.1.4 QMIX

QMIX is a Value Function Factorization algorithm that builds on the work of Value Decomposition Networks (VDN) (Sunehag, et al., 2018). The Centralized Learning in VDN is on the combined value function $Q_{tot}(\boldsymbol{\tau},\boldsymbol{a})$ – where $\boldsymbol{\tau}$ is the combined observation-action history as introduced in Section 2.2 – defined as

$$Q_{tot}(\boldsymbol{\tau},\boldsymbol{a}) = \sum_{i=1}^{n} Q_i(\tau_i, a_i; \theta_i). \tag{25}$$

The key idea is having the global argmax over the combined value function be the same as the combined individual argmax over the individual value functions.

$$\operatorname*{argmax}_{\boldsymbol{a}} Q_{tot}(\boldsymbol{\tau},\boldsymbol{a}) = \begin{pmatrix} \operatorname*{argmax}_{a_1} Q_1(\tau_1, a_1) \\ \vdots \\ \operatorname*{argmax}_{a_n} Q_n(\tau_n, a_n) \end{pmatrix} \tag{26}$$

QMIX (Rashid, et al., 2018) expands the family of monotonic Q functions that can satisfy the above constraint by enforcing the following on the value function Q_{tot} instead. This ensures all agents' Q-value functions contribute to Q_{tot}.

$$\frac{\partial Q_{tot}}{\partial Q_i} \geq 0, \forall i \in [n] \tag{27}$$

Each agent has its own Deep Recurrent Q-Network (DRQN) (Hausknecht & Stone, 2015) to represent its value function $Q_i(\tau_i, a_i)$, and a mixing network over the outputs of the individual functions to represent the combined value function Q_{tot}. States are used in a hypernetwork to produce the weights for the mentioned mixing network. The figure below illustrates the architecture used in QMIX.

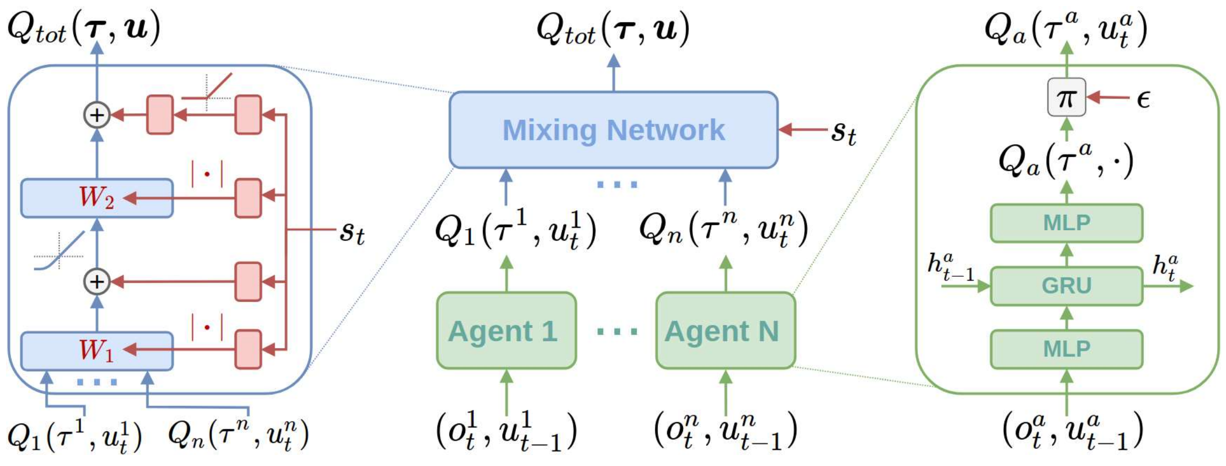

Figure 1: QMIX architecture (Rashid, et al., 2018)

4.1.5 DIAL

Reinforced Inter-Agent Learning (RIAL) (Foerster, Assael, de Freitas, & Whiteson, 2016) is a Learn to Communicate algorithm that has every agent observing their own local observation, with a limited bandwidth communication channel available to communicate with other agents. RIAL defines the messages space $\mathcal{M}$ and action space $\mathcal{A}$, which can be observed by other agents. The message space $\mathcal{M}$ works similary to the action space $\mathcal{A}$, where $m_i \in \mathcal{M}_i$ denotes a message from agent i, and $\boldsymbol{m} \in \mathcal{M}$ is the combined message vector from all agents. RIAL makes use of DRQNs to address partial observability. It also disables experience replay, to address experience becoming obsolete and misleading - the added communication aspect of the algorithm, combined with the use of DRQNs, non-stationarity and training together can cause newer policies, and thus their actions, to be vastly different than old ones.

Each agent i has their own Q-network at time t, taking in as input their local observation o_t^i, the RNN hidden state h_{t-1}^i, and the messages of other agents m_{t-1}^{-i}. The output of the Q-network makes use of an action selector with an ϵ-greedy policy to pick out $a_t^i \in \mathcal{A}$ and $m_t^i \in \mathcal{M}$. The architecture uses two Q-networks Q_a and Q_m to make sure there are linear $|\mathcal{A}| + |\mathcal{M}|$ outputs, maximising over $\mathcal{A}$, then $\mathcal{M}$, as opposed to quadratic $|\mathcal{A}||\mathcal{M}|$ outputs maximizing over $\mathcal{A} \times \mathcal{M}$. As such, the Q-network $Q(o_t^i, h_{t-1}^i, m_{t-1}^{-i}, a_t^i, m_t^i)$ is modelled via the given Q-networks, split into two different networks $Q_a(o_t^i, h_{t-1}^i, m_{t-1}^{-i}, a_t^i)$ and $Q_m(o_t^i, h_{t-1}^i, m_{t-1}^{-i}, m_t^i)$. The figure below illustrates the architecture of RIAL.

While there is an avenue for message passing with RIAL, and parameter sharing detailed in Section 4.3 can also improve the centralized learning of the algorithm, there is still a lack of feedback from agents on the communication actions they receive from other agents. Differentiable Inter-Agent Learning (DIAL) aims to address this by letting messages flow from agent to agent for richer feedback. In the centralized learning stage, the communication actions are instead replaced by direct connections between the output of one agent's network to the input of another agent. The other agents' messages m_{t-1}^{-i} then go through a discretize/regularize unit (DRU) before sending it to the input of each agent in the next time step. This is in contrast with the messages going into the action selector as in RIAL.

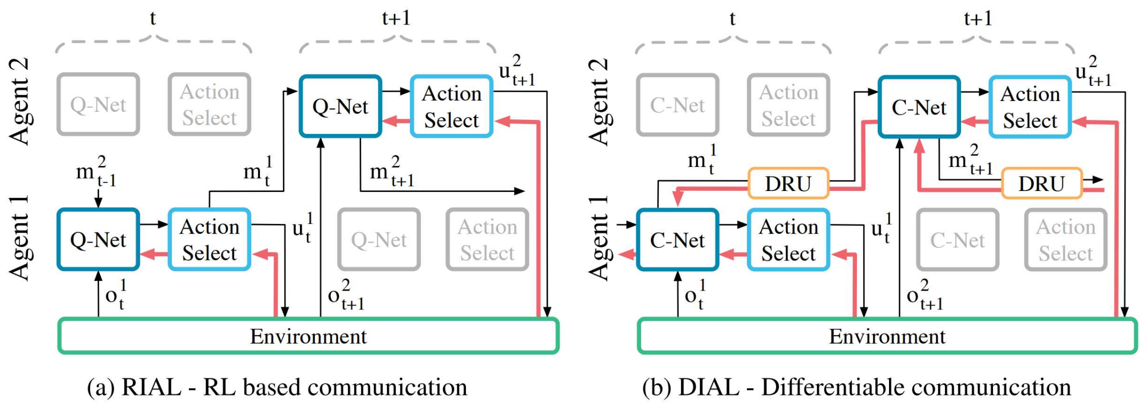

Figure 2: RIAL & DIAL (Foerster, Assael, de Freitas, & Whiteson, 2016)

4.1.6 SAF

The Stateful Active Facilitator (SAF) (Liu, et al., 2023) is a novel approach to Comm-MARL by adding a shared knowledge source (KS) to the established architecture of actor-critic algorithms as detailed previously. This shared KS will take in the observation signals of all agents and sift and interpret them before passing them to the critic. Information is thus refined by the KS, acting as an information bottleneck, implementing a more efficient form of centralization. The message space $\mathcal{M}$ and action space $\mathcal{A}$ can similarly be defined here. The generated message space $\mathcal{M}'$ can also be defined, different from the read message space $\mathcal{M}$. Here, messages $m' \in \mathcal{M}'$ are generated and stored in the KS, while messages $m \in \mathcal{M}$ are retrieved from reading from the KS. These messages can be different due to the cross-attention mechanism used to read from and write to the KS.

The cross-attention (CA) mechanism (Chen, Fan, & Panda, 2021) can be described using the query q, key k, value v, and their corresponding weight matrices W_q, W_k, W_v. Mathematically, with inputs x, x_c, CA can be expressed as

$$q = x_c W_q, k = xW_k, v = xW_v,$$
$$A = softmax\left(\frac{qk^T}{\sqrt{d}}\right), CA(x) = Av, \tag{28}$$

where $W_q, W_k, W_v \in \mathbb{R}^{C\times d}$ are the learnable weight matrix parameters.

SAF uses a four-step procedure to write to and read from the KS, detailed in the following paragraphs.

Step 1: Generating the Messages. For each observation from agent i at each time step t, o_t^i, a common encoder g_θ is applied, where d_m denotes the dimension of observations, and thus messages,

$$m'_{i,t} = g_\theta(o_{i,t}) \in \mathbb{R}^{d_m} \tag{29}$$

The set of messages generated at time step t are denoted by $\boldsymbol{m}'_t = \left(m'_{1,t}, m'_{1,t}, \ldots, m'_{n,t}\right)^T \in \mathcal{M}' \subseteq \mathbb{R}^{n\times d_m}$.

Step 2: Writing into the Knowledge Source (KS). Then, the encoded messages m' from each agent compete to write to the KS. The KS is represented as having L slots, as $F_t \in \mathbb{R}^{L\times d_l}$. Each of the L slots has dimension d_l. Messages will compete to write into each KS state slot with the cross-attention mechanism, which is a linear projection of the messages $\boldsymbol{m}'_t$ using a neural network. The query $\tilde{Q}$ is defined as a linear projection of F_t

$$\tilde{Q} = F_t \widetilde{W}^q. \tag{30}$$

The KS state can be updated as

$$F_t \leftarrow softmax\left(\frac{\tilde{Q}\left(\boldsymbol{m}'_t \widetilde{W}^e\right)^T}{\sqrt{d_e}}\right)\boldsymbol{m}'_t \widetilde{W}^v. \tag{31}$$

$\widetilde{W}^e, \widetilde{W}^v$ are the corresponding weight matrices that represent the linear projections of their vectors, and d_e is the column dimensionality of matrix $\widetilde{W}^e$. It can be seen that $A = softmax\left(\frac{\tilde{Q}(\boldsymbol{m}_t'\widetilde{W}^e)^T}{\sqrt{d_e}}\right)$ constitutes the cross-attention mechanism using the query $\tilde{Q}$ as a linear projection of the KS, F_t, and the key $k = \boldsymbol{m}_t'\widetilde{W}^e$, as well as value $v = \boldsymbol{m}_t'\widetilde{W}^v$, being linear projections of the messages $\boldsymbol{m}_t'$. Then, using a transformer encoder tower, made up of a perceiver encoder structure (Jaegle, et al., 2022), self-attention is applied to the KS.

Step 3: Reading from the KS. To read from the KS, cross-attention is once again used. Each agent creates a query from encoded partial observations $s_{i,t}$,

$$q_{i,t}^s = W_{read}^q s_{i,t} \in \mathbb{R}^{d_e}, \tag{32}$$

and all the queries are combined as $\boldsymbol{q}_t^s = \left(q_{1,t}^s, q_{2,t}^s, \dots, q_{n,t}^s\right)^T \in \mathbb{R}^{n \times d_e}$. The keys

$$\boldsymbol{\kappa} = F_T W^e \in \mathbb{R}^{l \times d_e} \tag{33}$$

are matched with the generated queries to produce the attention mechanism as

$$\boldsymbol{m}_t = softmax\left(\frac{\boldsymbol{q}_t^s \boldsymbol{\kappa}^T}{\sqrt{d_e}}\right) F_T W^v. \tag{34}$$

These read messages can produce a state with more information using the g_ϕ, parameterized as a neural network,

$$s_{i,t}^{SAF} = g_\phi\left(s_{i,t}, m_{i,t}\right). \tag{35}$$

$s_{i,t}^{SAF}$ is then passed to the critic to compute values. Since only the critic requires the use of $s_{i,t}^{SAF}$, and not the agent actors themselves, communication is only needed and learned during the training phase.

Step 4: Policy Selection. The paper also introduces the use of a policy pool to select policies for each agent, to reduce the reliance on assuming that agents in the environment are homogeneous, allowing for more heterogeneity of the agents. This contrasts with parameter sharing as detailed in Section 4.3, which have the actor policy networks be shared to improve performance with more homogeneous agents. To select a policy for each agent, the authors then have each agent select a policy from a policy pool. The input of a key initialized at the start of training, as well as an encoded partial observation $q_{i,t}^{policy} = g_{psel}(s_{i,t})$ are taken as input, producing an $index_i$ for which $\pi^{ind}\ \ {}_i$ is chosen for agent i.

It is worth noting that the implementation used in this paper, as well as in the results later in Section 5.1, drop the use of the policy pool. Environments discussed later in Section 4.2.1 also describe scenarios with highly homogeneous agents, and thus do not benefit from the policy pool.

4.2 Environments

Environments are a large component of evaluating the performance of various RL algorithms. Comparative studies for recent MARL algorithms are not very common (Papoudakis, Christianos, Schäfer, & Albrecht, 2021), and require continued development and maintenance as the plethora of algorithms produced grows every year. To focus the study in this project on the algorithms rather than the environments used, a well-maintained repository of environment simulators would be a preferable choice for benchmarking. This would allow some scalability in evaluating the various algorithms fairly. The work of developing a simulator from scratch contains many complexities depending on the environment that is being modeled, such as that of a robotic warehouse (Mahadevan, 2022). This section discusses the environments utilised in this project.

4.2.1 EPyMARL

The Extended Python MARL framework (EPyMARL)[1] is an extension of PyMARL by the Whiteson Research Lab (WhiRL)[2] (Samvelyan, et al., 2019), developed to better evaluate MARL algorithms. Nine MARL algorithms were empirically compared in various co-operative multi-agent tasks, with focus on providing fair comparisons by ensuring consistency between the algorithm implementations (Papoudakis, Christianos, Schäfer, & Albrecht, 2021). EPyMARL also introduced support for OpenAI Gym[3] environments, a toolkit largely used for RL research (Brockman, et al., 2016). This gives the potential for further ease of comparison with future environments that may be created. The Autonomous Agents Research Group (AARG) from the University of Edinburgh[4] maintains the environments listed here.

[1] https://github.com/uoe-agents/epymarl/
[2] https://github.com/oxwhirl/pymarl
[3] https://github.com/openai/gym
[4] https://agents.inf.ed.ac.uk/blog/epymarl/

The **Multi-Robot Warehouse environment (RWARE)**[5] simulates a grid-world warehouse where robots are required to deliver shelves to workstations and return them after delivery. The environment is a cooperative, partially observable environment with sparse rewards, where agents can only observe a 3×3 grid around them. Each robot has a discrete action space $A = \{Turn\ Left, Turn\ Right, Forward, Load, Unload\}$. The reward system is sparse, and is a value of 1 for successfully delivering the requested shelf to the workstation, and 0 otherwise. The sparse reward system can prove to be a difficult challenge for algorithms to find the optimal sequence of actions to achieve a greater reward.

The **Level-Based Foraging (LBF)**[6] environment simulates a grid-world where food items are scattered randomly. Agents must collect the food items, where both agents and items are assigned levels, and agents can collect the items if the sum of levels is greater than or equal to the item's levels. Each agent has a discrete action space $A = \{None, North, South, East, West, Load\}$. The reward system provides rewards each time a food is loaded, and no reward otherwise. All agents involved in the loading of the food will receive a normalized reward depending on their level, with the total value of all rewards over all foods present in the grid being equal to 1. The reward function for agent i is detailed as

$$r^i = \frac{FoodLevel \times AgentLevel}{\sum FoodLevels \sum LoadingAgentsLevel}. \tag{36}$$

The **PressurePlate**[7] environment is another grid-world environment, where the goal is to reach a treasure chest located behind locked doors. Each agent is assigned a pressure plate which they will have to stand on to unlock doors for other agents to go through. The discrete action space per agent is $A = \{Up, Down, Left, Right, No - op\}$. The reward system is independent for each agent and is the negative normalized Manhattan distance to their plate if they are in the same room as their plate, or the negative number of rooms to their plate otherwise. It can be noted that the reward for this environment is designed to be negative always.

The Figure below showcases illustrations of the mentioned environments.

[5] https://github.com/semitable/robotic-warehouse
[6] https://github.com/semitable/lb-foraging
[7] https://github.com/uoe-agents/pressureplate

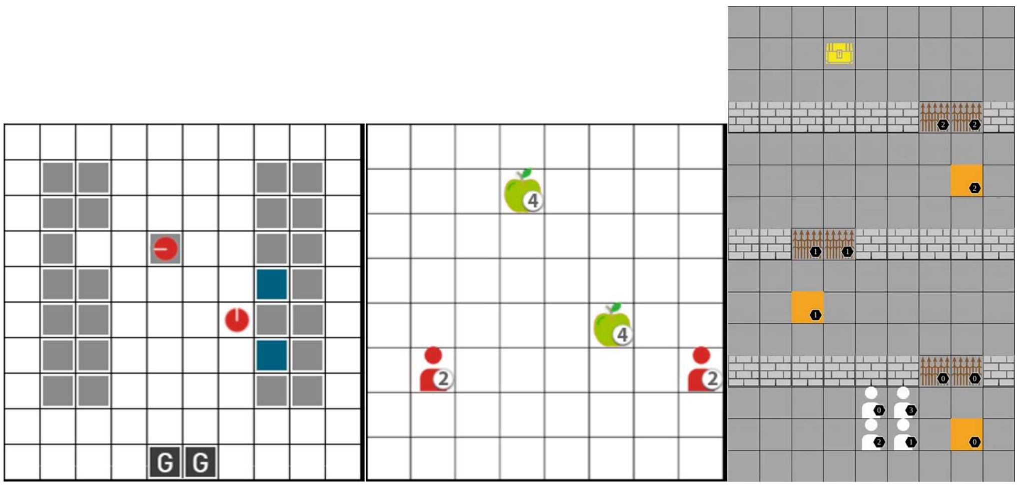

Figure 3: Environments
(a): Multi-Robot Warehouse (RWARE),
(b): Level-Based Foraging (LBF),
(c): PressurePlate
(Christianos, Schäfer, & Albrecht, 2020)

From the benchmark (Papoudakis, Christianos, Schäfer, & Albrecht, 2021), MAPPO does well in the RWARE environment, achieving the best results over all the algorithms. Its independent variant IPPO also performs competitively in the same environment. MAA2C performs competitively in various environments, and particularly does well in the LBF environment. IA2C also is close in performance to MAA2C in most cases, with the centralized value function squeezing a slight performance boost in MAA2C. QMIX achieves high returns across various tasks but may not be the best performing algorithm across all tasks. However, QMIX does not perform as well in the sparse reward environment of RWARE.

Tasks\Algs.	IPPO	MAPPO	MAA2C	QMIX
RWARE Tiny 4p	31.82 ± 10.71	$\mathbf{49.42 \pm 1.22}$	32.50 ± 9.79	0.30 ± 0.19
LBF 8x8-2p-2f-c	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	0.96 ± 0.07

Table 1: Benchmark Results (Papoudakis, Christianos, Schäfer, & Albrecht, 2021)

The table above shows maximum returns, with 95% confidence intervals, from running the algorithms on selected tasks of RWARE and LBF from the benchmark. In bold are the best performing algorithms for the task. The algorithms were run over five seeds in each task with parameter sharing, shown to improve performance, and detailed in Section 4.3. The description of the names of the tasks and the configuration they represent is detailed in Section 4.2.1.

The performance of SAF was not yet tested in this benchmark. The next few sections detail environmental concepts introduced by the authors of SAF, as well as its performance in those environments compared to the baseline algorithms.

4.2.2 Heterogeneity and Coordination

The authors of SAF from Section 4.1.6 define the concepts of heterogeneity and coordination as metrics that can be used to describe environments (Liu, et al., 2023), summarized as follows.

- Heterogeneity: Quantitative measure of the variation of environment dynamics
- Coordination: Quantitative measure of coordination required among agents to solve tasks

Formally, given an environment that has a collection of K different transition functions, $\{P(s_t, a_t) \mapsto s_{t+1}\}_{1\leq k\leq K}$, the next state of an agent at each time step t is governed by one of them, which is chosen by some latent variable v_t. K is then defined as the level of heterogeneity of the environment. If $K = 1$, then the environment is said to be homogeneous.

Coordination will then measure how much a subset of all agents $\mathcal{N}$ will need to work together to achieve the goal of the task. For a subset $\mathcal{G} \subseteq \mathcal{N}$ of $|\mathcal{G}| = k$ agents,

$$R_{\mathcal{G}}(\boldsymbol{s_t}, \boldsymbol{a_t}) = R_{\mathcal{G}}\left(s_{e,t}, \boldsymbol{s_t^{\mathcal{G}}}, \boldsymbol{a_t^{\mathcal{G}}}\right) \tag{37}$$

is defined as the reward for the subset of k agents are cooperating. $s_{e,t}$ is the state of the external environment, and $\boldsymbol{s_t^{\mathcal{G}}} = \{s_t^i\}_{i\in\mathcal{G}}, \boldsymbol{a_t^{\mathcal{G}}} = \{a_t^i\}_{i\in\mathcal{G}}$ are the states and actions of agents in the subset $\mathcal{G}$. The level of coordination c_t at each time step t is then the minimum size of a subset of agents $\mathcal{G}$ such that a positive reward can be obtained,

$$c_t = \min\{|\mathcal{G}|: \mathcal{G} \subseteq \mathcal{N}, R_{\mathcal{G}}(\boldsymbol{s_t}, \boldsymbol{a_t}) > 0\}. \tag{38}$$

The global coordination level of an environment can then be defined as

$$c = \max_t c_t. \tag{39}$$

For the environments detailed earlier, heterogeneity is defined to be $K = 1$ – the environments are homogeneous as there is only one transition function for the next states of agents to be governed by.

For the three environments mentioned earlier, the global coordination level can be obtained. In RWARE, it is possible for one robot to deliver all shelves, so $c = 1$. For LBF, the coordination depends on the maximum level of a food k and minimum level of an agent g, with the coordination level being bounded $c \leq k/g$. In PressurePlate, each agent at each time step either has their own plate to head toward, or they have to wait for the door to their room to open. Furthermore, every action at any time step results in a negative reward. As such, $c = 1$ here as well.

4.2.3 SAF in EPyMARL

As detailed earlier in Section 4.2.1, it is pertinent to make use of a proper benchmark for the fair comparison of the various algorithms. The SAF algorithm from Section 4.1.6 was released as open-source code by the authors[8], but make use of their own environments for basis of comparison.

Three cooperative MARL environments are introduced, and collectively called **HECOGrid** (Liu, et al., 2023). The environments follow a similar grid layout as those detailed in previous sections.

First, they introduce the **TeamTogether** environment, which bears close resemblance to the LBF environment detailed before. Agents must collect a treasure, which requires a certain number of agents to step on before it can be collected.

Then, there is the **TeamSupport** environment, which is an extension of TeamTogether. Here, an agent can collect the treasure if it has enough agents around the treasure within a certain fixed distance (set to 2 by default) which can support the collection.

Lastly, the **KeyForTreasure** environment extends TeamSupport further by requiring the collection of a key before being able to collect the treasure. The figure below showcases the three HECOGrid environments.

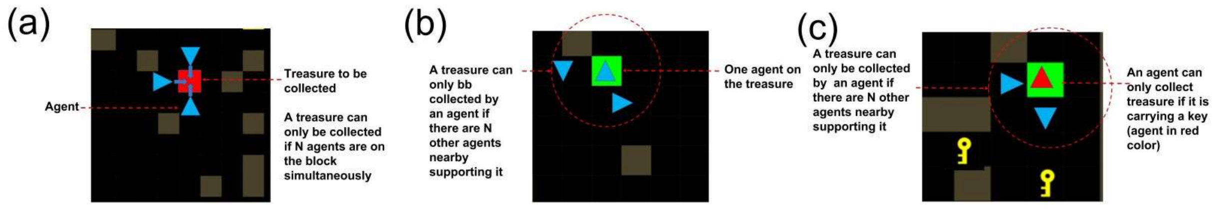
(a)
Agent
Treasure to be collected
A treasure can only be collected if N agents are on the block simultaneously
(b)
A treasure can only bb collected by an agent if there are N other agents nearby supporting it
One agent on the treasure
(c)
A treasure can only be collected by an agent if there are N other agents nearby supporting it
An agent can only collect treasure if it is carrying a key (agent in red color)

Figure 4: HECOGrid (Liu, et al., 2023)

[8] https://github.com/jaggbow/saf

The LBF and PressurePlate environments can be very closely compared to the HECOGrid environments, and can provide a proper benchmark for environments that SAF should thrive in. The results from the Liu, et al. (2023) show an improvement over MAPPO on the HECOGrid environments. However, since EPyMARL's environments are homogeneous, detailed in Section 4.2.2, heterogeneity, which is tested for by the SAF paper, is not tested in this paper.

To compare SAF with the existing baseline algorithms, we implement SAF within EPyMARL. SAF was implemented on top of MAPPO, while applying the key concept of SAF, to apply the transformation of states before sending the state to the centralized critic. This required an update of code for the critic module to insert the transformation. This code is also provided[9].

4.3 Parameter Sharing

Deep MARL algorithms commonly use two configurations during the training phase – parameter sharing, and no parameter sharing. With parameter sharing, all agents use the same parameters for their networks, while they each use their own set of parameters without it. Each actor and critic network would receive the identity of its agent as input to learn different behaviour. For Comm-MARL algorithms such as RIAL, the previous action messages a_{t-1}^{i}, m_{t-1}^{i} can also be input to the Q-network that pertains to the output action-message a_{t}^{i}, m_{t}^{i} (Foerster, Assael, de Freitas, & Whiteson, 2016). The shared parameters can then be optimized after calculating the loss function over all agents. Parameter sharing has been shown to improve performance of the MARL algorithms (Papoudakis, Christianos, Schäfer, & Albrecht, 2021), and should be used consistently to compare between each one.

[9] https://github.com/edu-ai/epymarl

5 Discussion

Here we discuss the results and challenges faced when running the algorithms on the environments mentioned.

5.1 Results

The algorithms were run with default configurations as given by the benchmarking repository. The training scripts were run on Python 3.8, PyTorch 1.13.0, OpenAI Gym 0.21.0. Other configurations of PyTorch and OpenAI Gym were found to lead to various dependency issues, seen in the next section.

All algorithms make use of learning rate values of $\alpha = 0.0005$, and discount factor of $\gamma = 0.99$.

MAA2C and MAPPO set the maximum $t_{max} = 20050000$, while QMIX uses the default $t_{max} = 2050000$. For all algorithms, their performance is tested every $t = 50000$ time steps. MAA2C and QMIX has `reward_standardisation = True`, while MAPPO sets this to `reward_standardisation = False`.

MAA2C and MAPPO use a soft policy selector for actions, taking a sample from $\pi_i(a_i|s_i;\theta_i)$, the distribution over actions given the observation. SAF on MAPPO and MAA2C follows the configuration of MAPPO and MAA2C respectively.

QMIX uses an epsilon-greedy scheme for exploration of actions, with start and finish values of $\epsilon = 1.0, 0.05$, linearly annealed over a time length of $T = 50000$. At each time step t_{env}, the probability that X, a random action is chosen, is

$$\Pr[X] = \max(0.05, 1.0 - 0.95 \times \frac{t_{env}}{50000}) \tag{40}$$

Otherwise $\underset{\boldsymbol{a}}{\operatorname{argmax}}\, Q_{tot}(\boldsymbol{\tau}, \boldsymbol{a})$ is chosen with probability $1 - \Pr[X]$.

SAF on MAPPO and MAA2C sets the number of slots in its KS to be $L = 2$. The results below show the number of agents being configured from two to four in the various tasks setup. This means that in each task, writing to the KS can require some competition for the agents, especially for configurations with more than 2 agents.

Test returns are then plotted for each algorithm-environment pair.

The table below details how the environment names are formatted for the various task configurations for each environment.

Environment	Environment Task Name	Name Format
RWARE	`rware-tiny-{Xag}-v1`	• `tiny` is the map size chosen, corresponding to one row, and three columns of groups of shelves. • Each group of shelves consists of 16 shelves arranged in an 8×2 grid. • `Xag` refers to the number of agents (i.e. `4ag` means 4 agents) in the environment. • Each agent can only observe a 3×3 grid centered around themselves.
LBF	`Foraging-{grid_size}x{grid_size}-{n_agents}p-{food}-v2`	• `grid_size` is the size of both the horizontal and vertical lengths of the environment. • `n_agents` is the number of agents. • `food` is the number of food items scattered in the environment. • Agents observe the coordinates of food in their observable radius, which is the entire grid, by default.
PressurePlate	`pressureplate-linear-{n}p-v0`	• `n` refers to the number of agents. • This work sets $n = 4$, which sets the room size to 15×9 grid by default.

Table 2: Environment Task Name Format

For all name formats, the version number at the end simply refers to the latest version available on their source repositories.

Although all algorithms were trained until the t_{max} values, the graphs may be truncated depending on when the maximum mean of test returns of all n algorithms occurs. Let G denote the set of algorithms for a graph, so $|G| = n$. For each time step $t \leq t_{max}$, if algorithm $g \in G$ outputs a test return of r_t^g, with the mean of returns for time step determined as

$$\overline{r_t} = \frac{1}{n} \sum_{g \in G} r_t^g. \tag{41}$$

The graph will then be truncated at the first index where

$$i = \underset{\mathrm{t}}{\operatorname{argmax}}\, \overline{r_t}. \tag{42}$$

In the following graphs, the titles detail the exact configuration used within each type of environment.

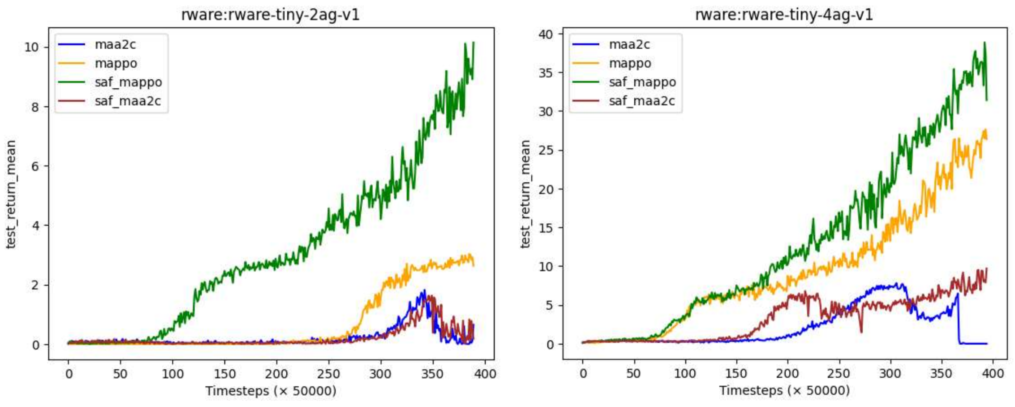

Figure 5: RWARE Test Returns

On RWARE, MAPPO, SAF on MAPPO, IPPO and MAA2C were evaluated, based on their high performance on the benchmark. Previously, MAPPO faced issues in training, as detailed in Section 5.2.1, and thus the performance of IPPO, its Independent Learner counterpart, was evaluated instead. However, this was resolved, and MAPPO performs the strongest of the algorithms coming from the EPyMARL benchmark, excluding the SAF augmentation. MAPPO provides the stronger performance against MAA2C. MADDPG was not successfully run due to the system used not meeting the minimum memory requirements. QMIX performs extremely poorly on the benchmark and was not tested here.

SAF on MAPPO produces the strongest results on the RWARE environment. This could be attributed to the partial observability of the environment, since agents within RWARE only have a small 3 × 3 grid centered on themselves, in which all entities can be observed. By leveraging on the KS available from the SAF algorithm, information on each agent's immediate surroundings may be used to contribute to the essential information for all agents in the grid. This information could be used to optimize for all agents movements to be better positioned to anticipate incoming delivery requests.

On the other hand, SAF does not improve the performance of MAA2C in all tasks tested. SAF may require the leveraging of MAPPO having on-policy optimization (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017) to take advantage of the extra information provided by its KS.

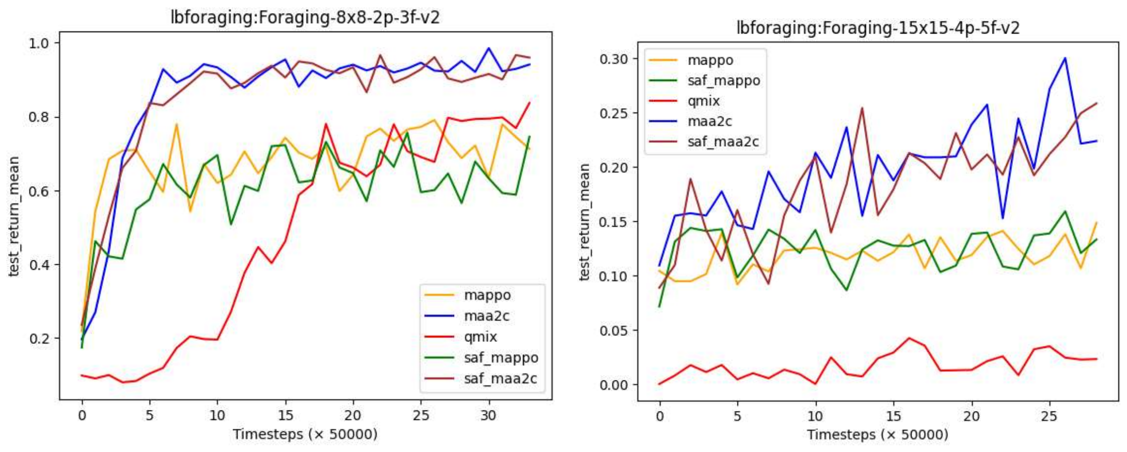

Figure 6: LBF Test Returns

On LBF, MAPPO, MAA2C, and QMIX perform competitively, achieving high returns close to the maximum reward of 1. This matches results given in the benchmark and shows the strong performance of the state-of-the-art in a simple, small, grid-world environment. In the smaller `8x8` grid, MAA2C performs the strongest, with MAPPO, SAF, and QMIX performing similarly right behind. In the larger `15x15` grid with 4 players, MAA2C still performs best, but QMIX now drops much lower in performance than MAPPO or SAF. SAF is not seen to improve the performance of MAPPO in this environment. With the small number of agents in the environment and a wider radius of observability, it is likely that having a KS does not provide as much advantage as it does in the slightly more complex RWARE environment.

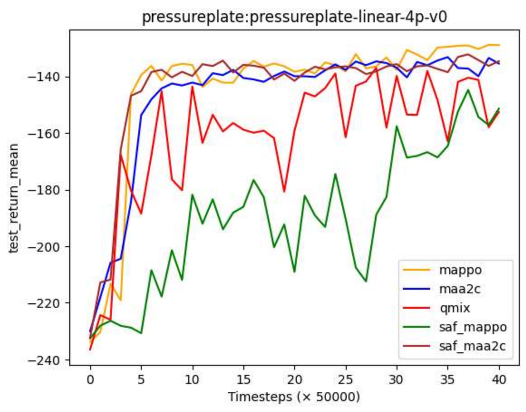

Figure 7: PressurePlate Test Returns

On PressurePlate, MAPPO, MAPPO with SAF, MAA2C, and QMIX perform competitively as well. This environment only has negative rewards before reaching the goal. MAPPO performs best on the default configurations. SAF decreases the performance of MAPPO slightly here. This could be attributed to the fact that the sharing of knowledge in this environment is not too useful for the agents. Since each agent in PressurePlate would have their own plate to take care of, there is not much coordination needed as their tasks are almost independent. Having more information from other agents could slow down learning the optimized path to each agent's plates. The figure below shows more specifically the difference in performance for MAPPO and MAPPO with SAF throughout training in the PressurePlate environment.

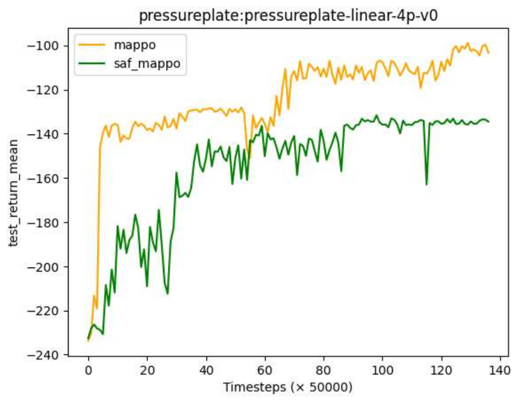

Figure 8: MAPPO/SAF on PressurePlate

In terms of real-world runtime, training a single algorithm at a high t_{max} value as configured can take over 12 hours on an NVIDIA A100 GPU. With the aid of the NUS School of Computing

(SoC) clusters and the Slurm workload manager, we can use the GPUs to run the various configurations in parallel.

5.2 Challenges

Training MARL algorithms can involve challenges and complexities that are almost guaranteed to occur. These can range from issues with the algorithms themselves, environments they are tested in, or simply with the compatibility of various dependencies required to run them.

5.2.1 Training Instability

While training MAPPO on the RWARE environment, an error occurs midway through where the actor network $\pi_i(a_i; \theta_i)$ ends up using invalid values for its parameters. More specifically, θ_i is made of `nan` values, which causes the action selector to fail in choosing a valid action. The figure below shows the returns of RWARE increasing until around 10000 episodes in, where there is a steep drop.

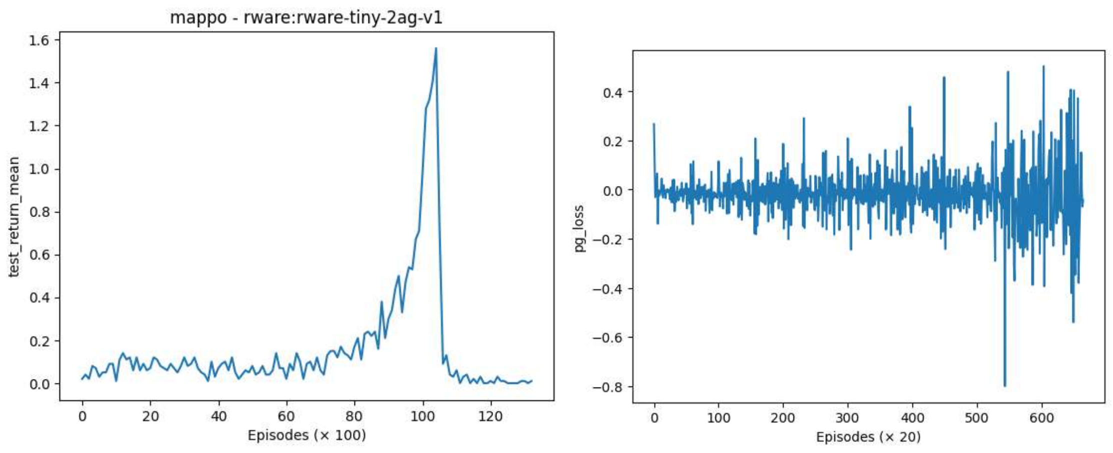

Figure 9: Misconfigured MAPPO on RWARE

Although the commands run were from the published source repositories, the error persists in various configurations tested. These include running using Python versions 3.7-3.9, with PyTorch versions 1.8-1.12, as well as earlier versions of the environments from their respective source repositories.

This showcases the sensitivity of training RL models. The model appears stable until just over 10000 episodes of training, where it diverges, ultimately leading to nan values propagating throughout the model parameters. This could be caused by an extremely high variance in the objective function, possibly requiring a tweak in hyperparameters. This issue has been reported to the maintainers of the source code repository[10], with a response to update a single line of their configuration. The configuration pertained to the standardization of their returns within the code, and managed to fix the issue, seen in Section 5.1.

5.2.2 Dependencies

There were incompatible packages that had to be accounted for when implementing the various algorithms and running them on the various environments. For example, while the latter two environments introduced, LBF and PressurePlate, are compatible with OpenAI Gym versions above 0.21.0, RWARE unfortunately is an exception. This issue is acknowledged by RWARE maintainers but has not seen a fix yet[11].

6 Conclusion

This project has shown that various MARL algorithms can be used and benchmarked together to form a baseline of performance. Various environments are introduced, as shown in the EPyMARL framework, to investigate the effectiveness of each algorithm. A focus on Comm-MARL algorithms was chosen to facilitate knowledge sharing in multi-agent environments, with the SAF mechanism being implemented in EPyMARL, on top of algorithms such as MAPPO and MAA2C, to better compare its ability to state-of-the-art implementations of the baseline algorithms. It has been shown to improve performance in at least one of the environments for the MAPPO algorithm, while leaving performance largely unchanged for MAA2C. The effectiveness of the SAF augmentation can be attributed to the mechanisms of the environments themselves, or certain advantages from the algorithm's attributes.

[10] https://github.com/uoe-agents/epymarl/issues/32
[11] https://github.com/semitable/robotic-warehouse/issues/10

7 Future Work

It is pertinent to check the generalization of each algorithm, as it may be the case that an algorithm is specialized to work well only on a select few environments.

The goal of evaluating the algorithms as shown in the previous chapter was to establish a baseline level of performance. With the baseline established, we incorporated effective mechanisms to introduce coordination among the agents to achieve better learning performance.

We also established a mechanism for knowledge sharing and transfer among the various agents operating in the multi agent environment, SAF, which was implemented in the EPyMARL framework for a proper comparison. This proves to be the focus of the work in this project.

With the availability of various algorithms as detailed in this project, there is a potential to combine the ideas from some of them to improve or address the weaknesses of one another. A clear example is how SAF was used to improve MAPPO, demonstrated in this work, to leverage on the use of a KS for communication via communication actions. SAF can possibly be implemented over other state-of-the-art algorithms such as MADDPG to bring performance improvements over the baseline algorithms.

The configuration of the SAF augmentation itself could also be used as an improvement area. For example, SAF uses its own fully connected neural networks to implement the mechanism, which makes use of hyperparameters such as number of hidden units, or number of layers, and activation functions. The number of KS slots could also be optimized for. As detailed in Section 4.1.6, the policy pool from the original paper was not implemented. This could also be a possible direction to consider, especially if looking to investigate more heterogeneous environments.

The various Comm-MARL algorithms can also be implemented and benchmarked together, similar to how the EPyMARL framework was created to provide a fair comparison. These communication actions that these algorithms utilize can be commonly abstracted into such a framework. Such a benchmark can help researchers focus on the areas of Comm-MARL that affect performance or reveal any issues in overfitting to certain environments.

Complexity analysis can also be applied to compare the efficiency of various algorithms. Much of the research on MARL can assume unlimited resources in terms of time and compute power, but certain environments can require the algorithm to be within a threshold, especially in the execution stage. This analysis can also lead to algorithms that can better scale to much larger number of agents that can still be feasibly trained and executed in reasonable time.

8 References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *arXiv*.

Chen, C.-F. (., Fan, Q., & Panda, R. (2021). CrossViT: Cross-Attention Multi-Scale Vision Transformer for Image Classification. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (pp. 357-366). Montreal: IEEE.

Christianos, F., Schäfer, L., & Albrecht, S. V. (2020). Shared experience actor-critic for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 10707-10717.

Claus, C., & Boutilier, C. (1998). The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 746–752). Madison: AAAI.

Foerster, J. N., Assael, Y. M., de Freitas, N., & Whiteson, S. (2016). Learning to Communicate with Deep Multi-Agent Reinforcement Learning. *Proceedings of the 30th Conference on Neural Information Processing Systems* (pp. 2145–2153). Barcelona: NIPS'16.

Hausknecht, M. J., & Stone, P. (2015). Deep Recurrent Q-Learning for Partially Observable MDPs. *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents.* Arlington: AAAI Press.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-term Memory. *Neural computation*, 1735-80.

Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., . . . Carreira, J. (2022). Perceiver IO: A General Architecture for Structured Inputs & Outputs. *International Conference on Learning Representations*.

Liu, D., Shah, V., Boussif, O., Meo, C., Goyal, A., Shu, T., . . . Bengio, Y. (2023). Stateful active facilitator: Coordination and Environmental Heterogeneity in Cooperative Multi-Agent Reinforcement Learning. *The Eleventh International Conference on Learning Representations.* Kigali: ICLR.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., & Mordatch, I. (2017). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *Proceedings of 31st Conference on Neural Information Processing Systems.* Long Beach: Curran Associates, Inc.

Lyu, X., Xiao, Y., Daley, B., & Amato, C. (2021). Contrasting Centralized and Decentralized Critics in Multi-Agent Reinforcement Learning. *Proceedings of the 20th International Conference on Autonomous Agents and multi-agent Systems* (pp. 844-852). Online: International Foundation for Autonomous Agents and Multiagent Systems.

Mahadevan, R. (2022). *Multi-agent strategy learning for warehouse robots.* B.Eng. Dissertation, Department Of Electrical & Computer Engineering: National University of Singapore.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., . . . Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. *Proceedings of the 33 rd International Conference on Machine.* New York: JMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., . . . Ku, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*, 529-533.

Ong, S. C., Png, S. W., Hsu, D., & Lee, W. S. (2009). POMDPs for Robotic Tasks with Mixed Observability. *Robotics: Science and Systems V.* Seattle: The MIT Press.

Oroojlooy, A., & Hajinezhad, D. (2019). A Review of Cooperative Multi-Agent Deep Learning. *CoRR*.

Papoudakis, G., Christianos, F., Schäfer, L., & Albrecht, S. V. (2021). Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms in Cooperative Tasks. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS).*

Rashid, T., Samvelyan, M., Witt, C. S., Farquhar, G., Foerster, J., & Whiteson, S. (2018). QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. *Proceedings of the 35th International Conference on Machine Learning* (pp. 4295-4304). Stockholm: PMLR.

Samvelyan, M., Rashid, T., Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., . . . Whiteson, S. (2019). The StarCraft Multi-Agent Challenge. *CoRR*.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). Trust Region Policy Optimization. *CoRR*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. *Proceedings of the 31st International Conference on Machine Learning* (pp. 387-395). Beijing: PMLR.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., . . . Graepel, T. (2018). Value-Decomposition Networks For Cooperative Multi-Agent Learning. *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems* (pp. 2085–2087). Stockholm: International Foundation for Autonomous Agents and Multiagent Systems.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* Cambridge: MIT Press.

Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Neural Information Processing Systems 12*, 1057-1063.

Wu, Y., Mansimov, E., Liao, S., Grosse, R., & Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. *Proceedings of the 31st Conference on Neural Information Processing Systems.* Long Beach: Curran Associates, Inc.

Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., & Wu, Y. (2021). The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games. *arXiv*.

Zhu, C., Dastani, M., & Wang, S. (2022). A Survey of Multi-Agent Reinforcement Learning with Communication. *ArXiv*.