Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Scaling MARL with Graph Neural Networks

Master's Thesis

Pius Kriemler

kpius@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Benjamin Estermann, Florian Grötschla
Prof. Dr. Roger Wattenhofer

March 17, 2024

Abstract

This thesis explores the application of Multi-Agent Reinforcement Learning (MARL) techniques to solve complex problems, drawing inspiration from natural swarms. Swarms, with their decentralised, autonomous agents operating on local information, exhibit remarkable collective behaviour that is scalable and resilient to perturbations. The use of MARL, especially in conjunction with Graph Neural Networks (GNNs), offers a promising approach to address modern computational challenges characterised by large data sets, distributed computing environments, and dynamic problem domains.

Our framework abstracts multi-agent environments as graphs and uses GNNs to facilitate communication and coordination between agents. Through locally sensed information and decentralised execution, scalability and resilience are achieved without constraints on the number or arrangement of agents. Using tools such as Ray and PyTorch Geometric, we demonstrate efficient policy learning for MARL scenarios. Experimental results highlight the adaptability and scalability of our approach, with promising performance on tasks such as transmission and moving.

Further investigation highlights challenges and opportunities for improvement. Sparse reward feedback in certain scenarios requires careful reward design, while attention mechanisms and the inclusion of absolute positions improve policy adaptability and information propagation. Future research directions include refining convolutional methods to better exploit edge attributes and quantifying the benefits of additional rounds of message passing.

Overall, this work contributes to the advancement of MARL techniques by providing scalable and robust solutions to complex computational challenges inspired by natural swarm behaviour.

Acknowledgment

I am profoundly grateful to Benjamin Estermann and Florian Grötschla for their invaluable guidance, patience, and unwavering support throughout the duration of this research. Their expertise and insightful critiques have not only shaped this thesis but have also profoundly influenced my growth as a scholar. I extend my deepest thanks for their mentorship.

To my family, words cannot express the gratitude I owe for your endless love, understanding, and encouragement. Your unyielding faith in my abilities has been a constant source of strength and motivation. This thesis stands as a testament to the sacrifices you have made for my education, and I am forever grateful for your support.

I also want to express my gratitude to the developers of ChatGPT and DeepL for their invaluable assistance in overcoming language barriers during the preparation of this work. Their powerful language models contributed to the clarity and effectiveness of my communication.

Lastly, I would like to extend my gratitude to everyone who directly or indirectly contributed to this research. Your wisdom, encouragement, and sometimes, even your challenges, have been crucial in completing this journey.

Thank you all for being part of this incredible journey.

Contents

Chapter 1

Introduction

Nature has long served as a source of inspiration for solving complex problems, particularly evident in the remarkable abilities of swarms. Consisting of large numbers of agents, swarms exhibit a remarkable ability to collectively solve challenging tasks without centralised control. From carefully building nests to efficiently gathering food or fighting off enemies, swarms demonstrate the power of collective action to overcome obstacles.

Each member of the swarm operates autonomously, relying solely on local information to guide its actions. The patterns and dynamics that ultimately lead to the successful completion of tasks emerge from the individual interactions within the collective. This approach is fully decentralized, without a global control center.

The swarm's scalability is significantly enhanced by the lack of global coordination efforts. Additionally, the swarm is not reliant on any specific individual, ensuring its resilience. Members can be added or removed dynamically, facilitating redundancy within the system. This redundancy, in turn, strengthens the swarm's resilience against failures or disruptions.

In computer science, it is essential to have scalable and robust algorithms to address the challenges posed by large and complex datasets, distributed computing environments, and dynamic problem domains. For instance, in the field of big data analytics, scalable algorithms are necessary to process and analyze massive volumes of data efficiently, enabling insights and discoveries that would otherwise be impractical or impossible to obtain. Similarly, in distributed systems and networking, robust algorithms are crucial for ensuring reliable communication, fault tolerance, and load balancing across multiple nodes or devices. These algorithms must handle varying workloads, network conditions, and failure scenarios while maintaining system performance and integrity. Having efficient, scalable, and resilient solutions is necessary to effectively meet the demands of modern applications and technologies.

Reinforcement Learning has been shown to be effective in deriving a strategy, called policy, from environmental feedback to solve problems that lack clear optimal approaches. Its effectiveness has been demonstrated in tackling complex challenges such as Chess [1] and Go [2]. Multi-Agent Reinforcement Learning extends this approach to scenarios where multiple agents collaborate to solve a problem collectively. Performing on super-human level in complex multi-agent games like Starcraft II [3] and Dota 2 [4] showcased its power, while the discovery of remarkable strategies within a task like Hide and Seek [5] made it accessible and engaging to the public through captivating visual representations.

In this thesis, we abstract multi-agent environments as graphs and employ Graph Neural Networks (GNN) [6, 7] to learn communication and coordination between agents. Our approach relies on locally sensed information within an agent's neighborhood to derive its actions, allowing for fully decentralised execution. Additionally, there are no limitations on the number of agents or their arrangement within the neighborhood. Scalability in terms of agents is ensured by relying on the efficient computation of actions through graph convolutions. The resulting framework builds on Ray [8, 9] and PyTorch Geometric [10] to facilitate an efficient and flexible implementation of policies, as well as to scale hyperparameter tuning of models to large-scale compute clusters. It offers policy learning for both Single-Agent Reinforcement Learning (SARL) and Multi-Agent Reinforcement Learning (MARL) scenarios. Our focus will mainly be on the latter.

The remainder of the thesis is organized as follows: Chapter 2 provides an introduction to the topic and related work. Chapter 3 offers background information on the work. Chapter 4 introduces the training setup, while Chapter 5 presents the conducted experiments and their results. Chapter 6 provides a conclusion and outlook on future work.

Chapter 2

Related Work

Learning effectively in MARL scenarios requires an understanding not only of the environment's dynamics but also of other participants intentions. The non-stationary nature of an agent's environment, influenced by other agents' evolving policies, renders the learning process unpredictable and unstable.

Modeling all agents as a single entity with a joint action space offers coordinated behaviors but presents scalability issues due to exponentially expanding action spaces with the number of agents [11]. Moreover, this approach heavily relies on extensive communication for execution, i.e. broadcasting all agents state on each step, posing challenges in real-world scenarios.

Facilitating active communication among agents fosters a mutual understanding of each other's policies, allowing agents to interpret each other's actions, thereby reducing unpredictability. Different approaches vary in terms of what, when, and to whom agents communicate, all of which contribute to the learning process. Sukhbaatar et al. [12] introduced the 'CommNet' architecture, which includes a central controller that enables communication among agents via continuous vectors. The iterative communication process aggregates communication vectors from neighboring agents in each iteration to facilitate the exchange of information. In their proposed model 'VAIN', Hoshen [13] enhanced CommNet by incorporating Interaction Networks [14]. They replaced the mean aggregation of neighborhood communication with an attentional interaction mechanism that scales the importance of communication between agents. Both approaches use a single neural network as the central controller, resulting in fully connected communication. This presents scalability challenges as the number of agents increases.

Another effective strategy to mitigate the nonstationarity is to enhance the critic in actor-critic methods during training. This approach, known as 'central critic,' provides the critic with access to all states during training while constraining the actor to local observations. This framework is often referred to as centralised training decentralised execution [11], which equips agents with sufficient information to tackle nonstationarity without requiring awareness of the global state during execution. Iqbal et al. [15] propose the Multi-Actor-

Attention-Critic, 'MAAC', algorithm. Each agent has its own centralized critic, which incorporates information from other agents through an attention mechanism. The specificity of this attention mechanism is that the agents query each other for information. From the critic, they derive the actor for the agent. As with previous models, its scalability is limited due to the independent critic for each agent.

Scalability is a key requirement, as a swarm extends the principles of MARL to a large scale. Jiang and Lu [16] propose 'ATOC', a communication model enabling an agent to establish temporary communication groups using an attention mechanism. These groups can only be formed among neighbours, which makes the communication approach effective for also large number of agents. Within a group, communication takes place through a bi-directional LSTM unit. This enables selective information sharing between agents. Agents who belong to multiple groups can enable inter-group communication by passing on information from one group to another in subsequent rounds of communication.

While this local communication approach effectively addresses scalability challenges, it introduces new complexities. In a swarm, an agent's neighborhood is highly dynamic, unlike traditional neural networks that rely on fixed-size consistently structured input data. Scarselli et al. [6, 7] pioneered the use of traditional neural networks to compute graph convolutions, leading to the development of Graph Neural Networks (GNN). By abstracting agents and their neighborhood relations as a graph, it becomes possible to compute embeddings for each graph node. These embeddings can then be utilized by regressors for predictions on a node level. GNNs are naturally permutation invariant to the graph topology. Moreover, GNNs leverage the principles of convolutional operations, such as local connectivity and parameter sharing, to efficiently handle large and potentially sparse graphs. Recent advancements demonstrate their efficacy in abstracting complex systems across various domains, including fraud detection [17], traffic prediction [18], cancer prediction [19], and various others [20, 21, 22].

Ryu et al. [23] and Sheng et al. [24] have both proposed hierarchical communication structures to facilitate more efficient communication patterns. They use GNNs to effectively implement actors and central critics over large graphs. In Ryu et al.'s 'HAMA' algorithm, agents are first clustered based on predefined groups. Each agent then computes multiple node embeddings, one for each group, using Graph Attention Networks. To produce the final output embedding, the agent combines these embeddings using another layer of attention. This mechanism accommodates both cooperative and competitive settings, allowing actions to be weighted in favour of or against different groups. Sheng et al.'s framework, 'LSC', builds on this concept by enabling adaptive group formation. After group formation, the proposed intra-group communication separates agents within a group into high-level and low-level agents. High-level agents first aggregate information within their group, then exchange it with other high-level agents, and finally distribute the embedded information to low-level agents within their group.

Our approach utilizes graph abstraction and employs GNNs for both the actor and a central critic. The neighbourhood of agents is encoded as a graph, with the relative distance between them used as an edge attribute. We compared the performance of using only relative distances to knowing their absolute positions. Furthermore, our research investigates communication and coordination strategies in various grid world environments and a continuous physics-based environment. A notable feature is our focus on, but is not limited to, using only one message passing round for our actor networks, meaning that agents can only incorporate information from their immediate neighbours.

We investigated the performance on tasks inspired by the work of Hüttenrauch et al. [25] on "building a communication network" and "communication link". In their work, agents aim to cover as much area as possible while maintaining connectivity in one scenario, and to establish a communication link between two points in another. Their focus was primarily on building the communication structure rather than the communication process itself, whereas we tried learning both aspects simultaneously. Additionally, we conducted benchmarking of our model's performance on one of Mordatch's Multi-Agent Particle Environment (MPE) tasks [26, 27]. The MPE consists of a set of tasks featuring a simplified physics engine where successful communication is essential for solving the challenges. This environment has been widely adopted by various authors, including Jiang et al. [16], Iqbal et al. [15], Ryu et al. [23], and Sheng et al. [24], to evaluate their architectures.

Chapter 3

Background

3.1 Reinforcement Learning

In reinforcement learning, agents interact with an environment in a self-learning way. They sample their actions from a policy based on their observations, iteratively generating sample trajectories. For each action, they are rewarded, where the reward function defines the notion of 'good' by mapping state-action pairs to a numerical value. These trajectories are then used to improve the policy. Exploring an environment in search of good states, and exploiting such states, are the two equally important parts of finding the optimal policy.

3.1.1 Multi-Agent Scenario

The multi-agent (MARL) scenario is defined as a Decentralized Partial Observable Markov Decision Process (*Dec*-POMDP) [28]. It is defined by the tuple $\langle N, \mathcal{S}, \mathbb{A}, \mathbb{O}, \mathcal{R}, T, \gamma \rangle$, where N is the number of agents, $\mathcal{S}$ the state space, $\mathbb{A} = \{\mathcal{A}^i\}_{i=1}^N$ the joint action space, $\mathbb{O} = \{\mathcal{O}^i\}_{i=1}^N$ the joint local observation space, $\mathcal{R} = \{R^i : \mathcal{S} \times \mathcal{A}^i \times \mathcal{S} \to \mathbb{R}\}_{i=1}^N$ the reward functions, $T : \mathcal{S} \times \mathbb{A} \times \mathcal{S} \to [0,1]$ the state transition function, and $\gamma \leq 1$ the discount factor. We define all agents to be homogeneous with shared parameters, using identical local observation spaces, action spaces, and policies. For readability, we omit the superscripts in $\mathcal{A}^i$, $\mathcal{O}^i$, and R^i, and simply refer to these spaces as $\mathcal{A}$, $\mathcal{O}$, and R, respectively.

At timestep t, all agents synchronously sample $a_t^i \in \mathcal{A}$, under their local observation $o_t^i \in \mathcal{O}$, from the policy $\pi_\theta : \mathcal{O} \times \mathcal{A} \to [0,1]$. The policies in this work are neural networks, the subscript θ denotes the weights and biases of the networks. Applying all actions to the environment results in a state transition $s_t \to s_{t+1}$. Each state-action pair receives feedback $r_t^i = R(s_t, a_t^i, s_{t+1})$ on how favourable the new state is. Iterating this process over a finite horizon of size $T \in \mathbb{Z}$ produces a trajectory $\tau = (s_0, a_0, s_1, a_1, .., a_{T-1}, s_{T-1})$, also called an episode or rollout.

3.1.2 Single-Agent Scenario

The single-agent (SARL) scenario is defined as a Markov Decision Process (MDP). It is described by the tuple $\langle \mathcal{S}, \mathbb{A}, \mathbb{O}, R, T, \gamma \rangle$, whereas $\mathcal{S}$ defines the state space, $\mathbb{A}$ the concatenation of action spaces $\{\mathcal{A}^i\}_{i=1}^N$, $\mathbb{O}$ the concatenation of local observation spaces $\{\mathcal{O}^i\}_{i=1}^N$, $R : \mathcal{S} \times \mathbb{A} \times \mathcal{S} \rightarrow \mathbb{R}$ the reward function, $T : \mathcal{S} \times \mathbb{A} \times \mathcal{S} \rightarrow [0, 1]$ the state transition function, and $\gamma \leq 1$ the discount factor.

In our use case, the SARL scenario can be interpreted as an abstraction from a node-based view to a graph-based view. Whereas in MARL agents work for their own sake and are judged for their own actions, in the SARL scenario agents still act for their own sake, but are judged as a whole. The action $a_t \in \mathbb{A}$ is sampled from $\Pi_\theta : \mathbb{O} \times \mathbb{A} \rightarrow [0, 1]$, while under the hood Π_θ uses π_θ several times to sample the actions for the agents. The feedback is given for the state of the whole graph by $r_t = R(s_t, a_t, s_{t+1})$.

3.1.3 Objective

Let $R(\tau) = \sum_{t=0}^{T-1} \gamma^t \sum_{i=1}^N r_t^i$ for the MARL scenario and $R(\tau) = \sum_{t=0}^{T-1} \gamma^t r_t$ for the SARL scenario by abusing the notation. The goal in both scenarios is to find a policy that maximises the expected return of the sampled actions,

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau)]. \tag{3.1}$$

3.2 Proximal Policy Optimization

Proximal Policy Optimisation (PPO) is a policy gradient method [29]. The idea behind PPO, and it's close predecessor Trust Region Policy Optimization (TRPO) [30], is to use importance sampling to increase sampling efficiency over Natural Gradient methods [31, 32]. Small policy updates are necessary for importance sampling to work properly; large updates lead to inaccurate gradient estimation, causing training instability and degraded performance. While TRPO enforces similarity through a KL divergence constraint and requires the computation of a Hessian for the gradient, PPO achieves the same by clipping the objective function, avoiding the need for expensive second-order terms. Although less precise, PPO has been empirically shown to perform reasonably well, even in MARL settings [33].

3.2.1 Policy Gradient Methods

Policy gradient methods tune a policy by directly adjusting its parameters θ, often the weights and biases of the underlying neural network that implements the policy. Given the objective function, the gradients are efficiently computed by autograd [34], PyTorch's auto-differentiation software. The listing 3.1 shows the general procedure in pseudocode:

```
while objective $L$ not satisfied:
    collect trajectories $\tau \sim \pi_\theta$
    compute policy gradient $\nabla_\theta L(\theta)$
    update policy parameters $\theta_{new} = \theta + \alpha \nabla_\theta L(\theta)$
```

Listing 3.1: Policy Gradient Algorithm Pseudocode

A fundamental policy gradient can be derived directly from the objective in equation 3.1. Rewritten in terms of its parameters θ, the objective is

$$L^{PG}(\theta) = \hat{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\log \pi_\theta(a_t|s_t) R(\tau) \right], \tag{3.2}$$

where $a_t, s_t \in \tau$ and $\hat{\mathbb{E}}$ is the empirical mean over τ.

3.2.2 Actor-Critic

Estimating the expected return using Monte Carlo sampling introduces high variance due to the stochastic nature of policy and the environment. Parameterised baselines, called critics, are subtracted from this expected return to reduce variance and improve convergence [35].

A common baseline is the value function $V : \mathcal{S} \to \mathbb{R}$:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim \pi} \left[R(\tau) \mid s_0 = s \right]. \tag{3.3}$$

Using the advantage function $A : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as a baseline has been empirically shown to further improve training performance [36].

$$\begin{aligned} Q^{\pi}\left(s_t, a_t\right) &= \mathbb{E}\left[R(s_t, a_t) \mid s_t = s, a_t = a\right] && (3.4) \\ A^{\pi}\left(s_t, a_t\right) &= Q^{\pi}\left(s_t, a_t\right) - V^{\pi}\left(s_t\right) && (3.5) \\ &= R(s_t, a_t) + \gamma V^{\pi}\left(s_{t+1}\right) - V^{\pi}\left(s_t\right) && (3.6) \end{aligned}$$

The objective in the equation 3.2 using Advantage Actor-Critic is

$$L^{A2C}(\theta) = \mathop{\hat{\mathbb{E}}}_{\tau \sim \pi_\theta} \left[\log \pi_\theta(a_t|s_t) A^{\pi}\left(s_t, a_t\right) \right]. \tag{3.7}$$

Advantages in this work are computed using Generalized Advantage Estimates [37].

3.2.3 Clipped Surrogate Objective

The PPO actor uses the following Clipped Surrogate Objective:

$$L^{CLIP}(\theta, a_t, s_t) = \min\left(w_t A^{\pi_\theta}\left(s_t, a_t\right), \operatorname{clip}\left(w_t, \epsilon\right) A^{\pi_\theta}\left(s_t, a_t\right)\right) \tag{3.8}$$

where $w_t = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{old}}(a_t \mid s_t)$ is the importance sampling weight, $\operatorname{clip}(w, \epsilon) \in [1-\epsilon, 1+\epsilon]$ clips w to fall in the range of $1 \pm \epsilon$, and ϵ is the clip parameter.

The importance sampling weight is used as a measure of divergence between the old and new policies. This is natural as it is, in theory [38], a correction factor that tells how representative the samples of $\pi_{\theta_{old}}$ are when judged against π_θ. The complete PPO objective incorporates in addition the mean squared error value loss L^{VF} of the critic, alongside an entropy bonus S_{π_θ} to incentivise exploration:

$$L^{PPO}(\theta) = \mathop{\hat{\mathbb{E}}}_{\tau \sim \pi_{\theta_{old}}} \left[L^{CLIP}(\theta, s_t, a_t) - c_1 L^{VF}(\theta, s_t, a_t) + c_2 S^{\pi_\theta}\left(s_t\right) \right], \tag{3.9}$$

whereas $c_1, c_2 \in \mathbb{R}$ [29].

3.3 Graph Neural Networks

A graph $G = (V, E)$ is defined by a set of nodes V and a set of edges E. We define the state of a node $v \in V$ to be the feature vector $x^v \in \mathbb{R}^c$, and the state of the edge $(v, w) \in E$ for $v, w \in V$ to be the edge attribute $x^{vw} \in \mathbb{R}^{c'}$. Let $N(v) \subseteq V \times E$ be the neighbourhood function describing the set of all nodes and edges connected to v. GNNs compute the node embedding $h^v \in \mathbb{R}^m$ using node and edge states in $N(v)$ with Message Passing Neural Networks [39]. By aggregating the node embeddings with a permutation invariant aggregation function, it is also possible to compute an embedding h^G for the whole graph. The embeddings can be used for regression or classification of nodes or the whole graph.

Node embeddings are computed iteratively, with subscripts used to denote the message passing round. Edge embeddings are encoded only once. Initial embeddings are computed by encoding the states as follows:

$$h^{vw} = ENCODER_{edge}(x^{vw}) \quad \forall (v, w) \in E \tag{3.10}$$

$$h_0^v = ENCODER_{node}(x^v) \quad \forall v \in V. \tag{3.11}$$

In each round, neighbours compute a message with their edge and node states using $f^1 : \mathbb{R}^m \times \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m$. The messages are aggregated by a permutation invariant aggregation function $\bigoplus$, and combined with the current embedding by some function $f^2 : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m$. A round of message passing looks like this:

$$msg_{t+1}^v = \bigoplus_{w \in N(v)} f_t^1 \left(h_t^v, h_t^w, h^{vw}\right) \tag{3.12}$$

$$h_{t+1}^v = f_t^2 \left(h_t^v, msg_{t+1}^v\right). \tag{3.13}$$

Let K be the total number of rounds of message passing. The final output is computed by decoding h_K^v, or the aggregation of all h_K^v in the case of the whole graph, as such:

$$y^v = DECODER(h_K^v) \tag{3.14}$$

$$y^G = DECODER(\bigotimes(\{h_K^v | \forall v \in V\}), \tag{3.15}$$

where $\bigotimes$ is a permutation invariant aggregation function. Note that the final output of K-round GNNs can contain information from up to K hop neighbourhoods of v. We also denote the edge attribute x^{vw} as the relative distance between nodes v and w.

3.3.1 Convolutions

We experimented with the convolutions GINE [40], GATv2 [41] and Transformer [42] from the PyG library. GINE was chosen as the baseline, as it includes all neighbours equally. GATv2 was chosen to allow a more selective inclusion of neighbours through the attention weights. As attention proved to be beneficial, we also considered Transformer, as it places more emphasis on the edge attributes in the embedding computation, rather than just using it for the attention weight computation.

GINE

$$h_{t+1}^{v} = \mathbf{\Theta}_{MLP}\left((1+\epsilon)\cdot h_t^v + \sum_{w\in\mathcal{N}(v)} \mathrm{ReLU}\left(h_t^w + h^{wv}\right)\right), \tag{3.16}$$

whereas $\mathbf{\Theta}_{MLP}$ is a muli-layered perceptron and ϵ a trainable parameter.

GATv2

$$h_{t+1}^{v} = \alpha_{v,v}\mathbf{\Theta}_s h_t^v + \sum_{w\in\mathcal{N}(v)} \alpha_{v,w}\mathbf{\Theta}_t h_t^w, \tag{3.17}$$

$$\alpha_{v,w} = \mathrm{softmax}\left(\mathbf{a}^\top \mathrm{LeakyReLU}\left(\mathbf{\Theta}_s h_t^v + \mathbf{\Theta}_t h_t^w + \mathbf{\Theta}_e h^{vw}\right)\right), \tag{3.18}$$

whereas $\mathbf{\Theta}_s, \mathbf{\Theta}_t, \mathbf{\Theta}_e$ are trainable linear transformations.

Transformer

$$h_{t+1}^{v} = \mathbf{\Theta}_1 h_t^v + \sum_{w\in\mathcal{N}(v)} \alpha_{v,w}\left(\mathbf{\Theta}_2 h_t^w + \mathbf{\Theta}_6 h^{vw}\right), \tag{3.19}$$

$$\alpha_{v,w} = \mathrm{softmax}\left(\frac{\left(\mathbf{\Theta}_3 h_t^v\right)^\top \left(\mathbf{\Theta}_4 h_t^w + \mathbf{\Theta}_6 h^{vw}\right)}{\sqrt{|N(v)|}}\right), \tag{3.20}$$

whereas $\mathbf{\Theta}_{1..6}$ are trainable linear transformations.

3.4 Ray

Our implementation is built using the Ray ecosystem, mainly Ray Tune [9] and RLlib [8]. Ray Tune autonomously creates trials and allocates cluster resources to them to scale hyperparameter tuning. Each trial is given a parameter set with which it instantiates PPO, a torch module with the actor and critic networks, and an RLlib environment. Parameter sampling and trial generation is controlled by an ASHA scheduler [43] that uses early stopping to first efficiently find promising parameter sets and then tries to exploit them.

RLlib is a large and flexible open-source framework for Reinforcement Learning that supports SARL and MARL scenarios, and natively works with Ray Tune. More lightweight frameworks like rlpyt [44] and the on OpenAI's Baselines [45] based StableBaselines [46] offer little to no support for MARL applications. While being lightweight and providing MARL support, (e)pymarl [47, 48] provides no distributed training setup.

3.5 MESA

To implement the environmental dynamics, we used the agent-based modelling framework MESA [49]. It provides the functionality to create, manipulate and visualise agents in grid-based and continuous environments.

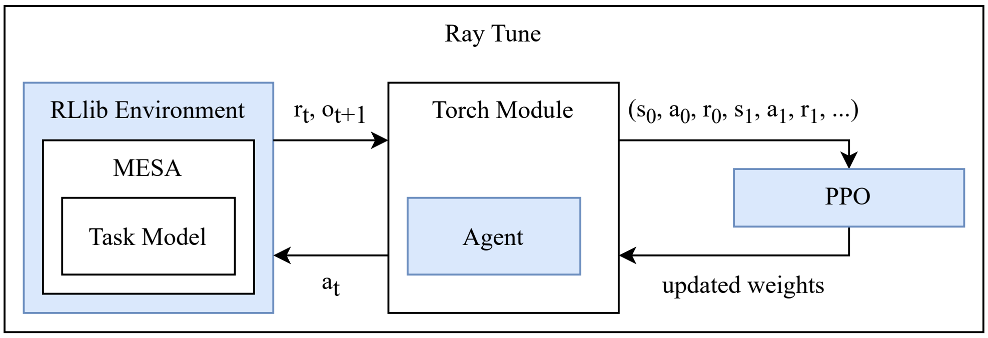


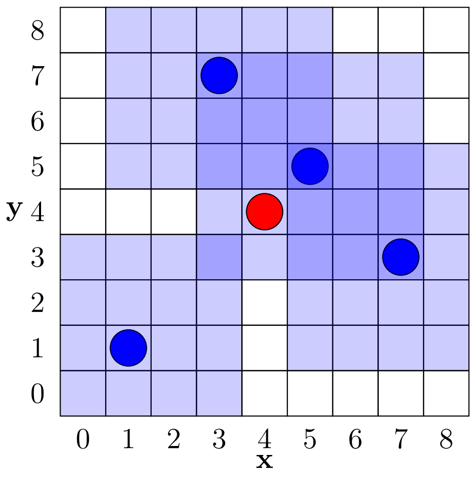
Figure 3.1: Reinforcement learning cycle. In blue, the three main components, the agent that interacts with the environment, while PPO improves the policy based on the sampled trajectories.

Chapter 4

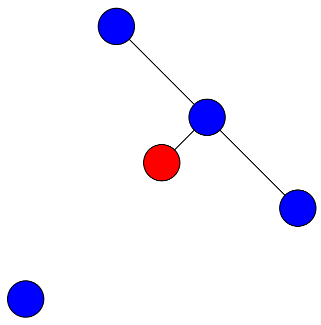
Method

4.1 Graph Abstraction

For the tasks presented, there are two types of agents: active and passive. The active agents are referred to as 'workers', who walk around, communicate, and try to coordinate. On the other hand, the passive agents are static objects in the environment. These objects could be oracles that provide information to the workers, meeting points, or walls.



(a) Example state: The red circle represents a passive agent, blue circles represent workers, and opaque blue squares represent the visible neighbourhood of workers.



(b) Graph representation: Agents that are in each other's neighbourhood are connected by an edge.

Figure 4.1: Abstracting a grid-world state to a graph.

Each agent is represented as a node in the graph, with edges connecting agents that are visible to each other. Visibility depends on the task at hand. The graph abstraction can be made in both discrete and continuous space. Unless otherwise stated, we assume that the environment is a grid-world, and we define agents to be visible to each other if they have a Chebyshev distance less than 3, e.g. they are at most 2 squares apart in either direction. By integrating static objects as nodes in the graph, we increase our flexibility in designing environments. This approach eliminates the need to encode static objects in the node state of workers, as they are inherently encoded in the structure of the graph. The process of constructing the graph using a grid-world environment and the default neighborhood function is illustrated in Figure 4.1.

4.2 Structured Observation Space

Ray Tune uses the interface implemented by RLlib environments to automate experiment generation, i.e. the creation and setup of parameterised environments and Torch modules. The definition of action and observation spaces in the environment specifies the allowed actions and observations, as well as their corresponding shapes. This definition also allows for the automatic instantiation of correctly sized networks and sampling spaces within the Torch module. It should be noted that Ray Tune restricts changes to the dimensions of these spaces throughout a trial.

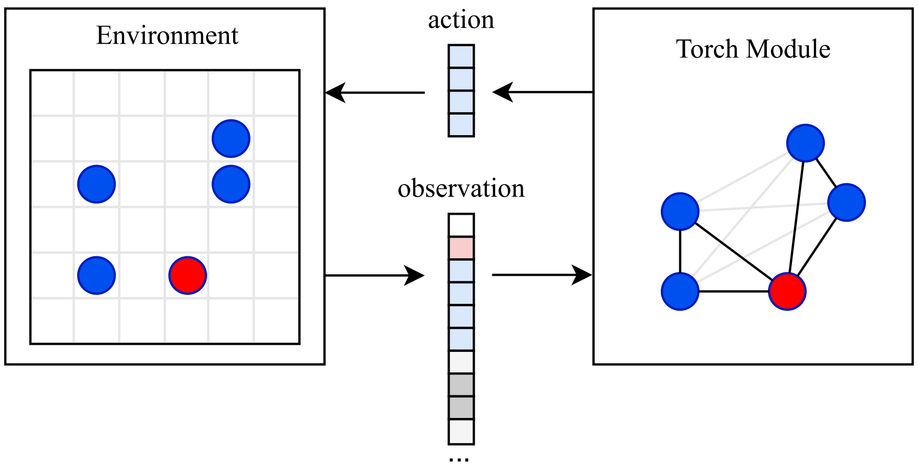


Figure 4.2: Communication between the environment and the Torch module in the Ray Tune workflow.

During training, the state of the environment at each time step must match the dimensions of the observation space when it is passed to the Torch module. To be able to reconstruct any observed topology, we encode the environment state using a specially structured observation space. This design distinguishes between graph topology and node state spaces. It also allows experimentation with node/edge state spaces without interrupting the basic functionality of graph construction. As can be seen in Figure 4.3, each observation consists of a hash and lists of subspaces for each node and possible edge. The boolean $flag$ within an edge entry allows to indicate the existence of the corresponding edge. Connections between nodes can thus be detected in the environment and passed to the Torch module without breaking the modularisation of the whole Ray Tune workflow. The $hash$ and node $flag$ are only used for optimisation purposes in the MARL scenario.

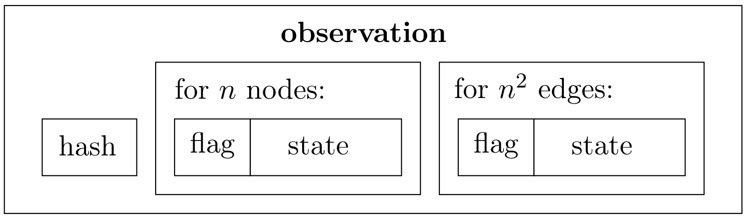


Figure 4.3: General structure of an observation.

4.3 Actor-Critic Networks

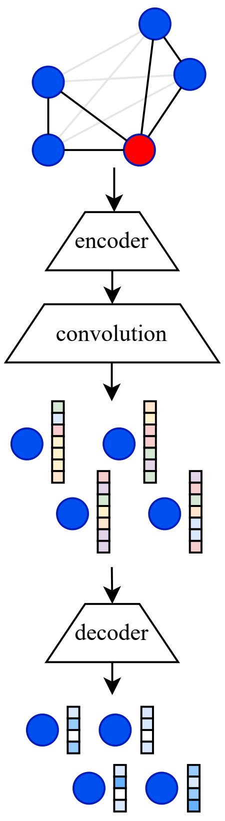
The actor network computes the actions of the agents based on the states of the agents in their neighbourhood. The critic network computes the values to calculate the advantage for the clipped surrogate objective in equation 3.8. Referring to section 3.3, y^v computed by the actor describes agent v's next action, while computed by the critic it defines the value (or the value y^G of the whole graph) with which it's advantage is calculated.

There is a difference in the granularity of a trajectory sample between the MARL and SARL scenarios. In the MARL scenario, the samples are from the perspective of the agent, i.e. each agent samples its own action. Meanwhile, the critic computes the value estimate for each worker individually. This setup ensures that each agent receives direct feedback tailored to its own state and actions. Conversely, in the SARL scenario, the samples are viewed from the graph perspective. Although the computation under the hood remains largely the same, a sample consists of the actions of all agents. Consequently, the critic computes only one value estimate, rather than individual values for each worker, providing a coarser feedback on the overall state and topology of the graph.

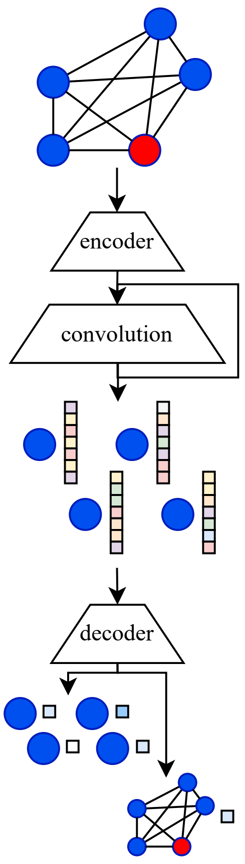
The actor and the critic have their own encoder, convolution and decoder networks. The encoder and decoder networks are fully connected linear layers, while the encoding size m is a tuned hyperparameter. Hyperparameter tuning is described in more detail in the 5.1 section.

We restrict the actor to perform only one round of message passing, as we want the actions to be computed only based on information from its actual neighbours.

The critic is free to perform up to 8 rounds of communication, where K is a tuned hyperparameter. Furthermore, the critic is centralised and computes it's output on the fully connected graph. This simulates the sharing of the global state, allowing each agent to consider the information of all other agents when estimating its advantage.



(a) Actor: Computes an agent's action from information from its neighbours in the graph in a message-passing round.



(b) Critic: Computes the value output of a node (or of the whole graph) from the fully connected graph in multiple message passing rounds.

Figure 4.4: Illustration of actor and critic networks, and computation of actions and values from the observed graph state.

4.4 Tasks

4.4.1 Lever Pulling

The lever-pulling task, originally introduced by Sukhbaatar et al. [12], serves to validate agents' communication skills. We recreated the task within our environment, maintaining its essential setup. 5 out of 500 workers are selected to be placed in the grid-world on the same square. This setup results in a fully connected graph where all edge features, representing the distance between any two workers, is zero. The workers' goal is to synchronously select five distinct levers.

4.4.2 Transmission

The transmission task investigates the fundamental information propagation capabilities among agents. A group of stationary workers is positioned around an oracle, which chooses a number for the workers to replicate. This number is referred to as the output, denoted as *out*, whereas $out \in \{0, 1, 2, 3, 4\}$. Workers are positioned in such a way that each has a communication link to the oracle, either through an edge to the oracle or through intermediate workers that act as transmitters. This arrangement necessitates that agents actively relay information to one another to solve the task. Each agent's state is equipped with its type, current output, and a hidden vector, while the oracle's hidden vector is always 0. The type allows workers to differentiate between other workers and the oracle.

4.4.3 Transmission Extended

The worker's state is augmented by including its relative position *pos* with respect to the oracle. This adjustment enables investigation into the influence of knowing absolute positions in the agent state versus solely having knowledge of local relative positions through the edge attributes.

4.4.4 Moving

Expanding on the concept of Transmission, movement is incorporated into the system. In addition to replicating and sharing the oracles output, workers are required to navigate the environment. Workers were given the capability to move one square horizontally and one square vertically in any direction per time step. At the start of each episode, workers were positioned to form a communication line, but tended to disconnect during early training, quickly resulting in a fragmented graph. To prevent excessive movement during the initial stages of learning, we employed this discretization techniques, making it more probable for agents to stay idle when dx and dy are sampled randomly. To discretize the movement to $dmov$, the actual movement the worker is going to perform, we follow the procedure outlined for the horizontal movement, and similarly for the vertical movement.

$$\text{movement on } x\text{-axis} = \begin{cases} -1 & \text{if } dx < -0.6 \\ 1 & \text{if } dx > 0.6 \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

4.4.5 Moving History

Drawing a parallel to the Transmission Extended task, we augmented the worker state by adding a global reference point. This reference point is determined by taking into account not only the worker's current relative location to the oracle but also their past three locations and movements. This enhancement is justified by drawing an analogy to natural agents. Natural agents often rely on a central base, such as a hive or den.

4.4.6 MPE Spread

We replicated the 'spread' task from the Multi-Agent Particle Environment, where n workers aim to approach n landmarks without colliding with each other. The neighbourhood function is fixed to always include the three closest workers and landmarks, regardless of spatial distance. The landmarks remain stationary while the workers move in a continuous space. The state of the workers consists of their absolute position and velocity. Their action is a force vector that affects their velocity through a simple physics engine. Most of the implementation was copied from the official PettingZoo implementation [50], while only changes were made to make the task compatible with our structured observation space.

4.4.7 MPE Spread Memory

The worker's state is supplemented by a hidden vector, as seen in previous tasks. This allows the worker to retain information from previous interactions.

4.5 Reward Functions

Will from now on refer to all networks as a model, e.g. the networks of actor and critic, encoder, decoder, etc., which together can solve a task.

Task name	**Node state space**	**Action space**
Lever Pulling	id_{agent}	id_{lever}
Transmission	$type$, out^t, $hvec^t$	out^{t+1}, $hvec^{t+1}$
Transmission Extended	$+\ pos^t$	out^{t+1}, $hvec^{t+1}$
Moving	$type$, out^t, $hvec^t$	out^{t+1}, $hvec^{t+1}$, $cmov$
Moving History	$+\ \{pos^i\}_{i=t}^{t-3}$, $\{dmov^i\}_{i=t-1}^{t-3}$	out^{t+1}, $hvec^{t+1}$, $cmov$
MPE Spread	$type$, pos^t, vel^t	$force^{t+1}$
MPE Spread Memory	$+\ hvec^t$	$force^{t+1}$, $hvec^{t+1}$

Table 4.1: Node state spaces and action spaces of the different tasks

Subspace	**Domain**
id	$\mathbb{Z}^+$
$type$	$\mathbb{Z}^+$
out	$\mathbb{Z}^+$
$hvec$	$[0,1]^m$
pos	$\mathbb{Z}^2$
$cmov$	$[-1,1]^2$
$dmov$	$\{-1,0,1\}^2$
vel	$[-1,1]^2$
$force$	$[-1,1]^2$

Table 4.2: Subspace domains

The models for Lever Pulling were trained using a shared reward, calculated as the number of different levers pulled divided by the total number of levers. The same reward was used to evaluate the trials.

The models for MPE Spread were trained using the reward function replicated from [23]. Namely, each worker was rewarded with the distance to the nearest landmark with an additional collision penalty of -1 for each collision. The same reward was used to evaluate the trials.

The models for Transmission, Transmission Extended, Moving, and Moving History were trained using the reward functions outlined in tables 4.3 and 4.4. We define the output of the worker v as out_v and the output of the oracle as out_o. The correctness of the worker's output is determined by comparing it to the oracle's output, i.e., whether $out_v = out_o$. The number of workers with correct outputs is represented by $outs_{correct}$, while the number of workers with incorrect outputs is represented by $outs_{wrong} = n - outs_{correct}$. The distance measure, $dist_v$, is defined as the Chebyshev distance, i.e. if the positions of the worker and the oracle are (x_v, y_v) and (x_o, y_o) respectively, $dist_v = \max(\text{abs}(x_o - x_v), \text{abs}(y_o - y_v))$. The boolean variable $connected_v$ indicates the existence of a communication line between the worker and the oracle, represented by a path in the graph from the oracle to the worker. For the evaluation, we used the fraction of correct outputs as a metric, as replicating the oracles output is the main objective.

Name	**Reward**	
random	no training	
shared binary	if $outs_{correct} = n$:	$r = 1$
	otherwise:	$r = -1$
shared sum	if $outs_{correct} = n$:	$r = n$
	otherwise:	$r = -outs_{wrong}$

Table 4.3: Description of the reward functions used in SARL scenarios

4.6 Curriculum Training

The models for Transmission, Transmission Extended, Moving and Moving History were trained using curriculum learning.

The main variable that changed was the placement of the workers at the beginning of each episode. Let us first define the different placement strategies. The 'unidirectional' placement always placed the workers in a straight line to the right of the oracle. The 'multidirectional' strategy places workers in a straight line extending from the oracle in any direction - up, down, left or right. The 'randomised multidirectional' strategy also randomly displaced workers perpendicular to the main placement direction. In all these strategies, the workers always formed a

Name	**Reward**	
random	no training	
shared binary	if $outs_{correct} = n$:	$r_v = 1$
	otherwise:	$r_v = -1$
shared sum	if $outs_{correct} = n$:	$r_v = n$
	otherwise:	$r_v = -outs_{wrong}$
individual	if $out_v = out_o$:	$r_v = 1$
	otherwise:	$r_v = -1$
spread	if $out_v = out_o$:	$r_v = dist_v$
	otherwise:	$r_v = -1$
spread (connected)	if $out_v = out_o$ and $connected_v$:	$r_v = dist_v$
	elif $out_v = out_o$:	$r_v = 0.25 \cdot dist_v$
	elif $connected_v$:	$r_v = -0.5$
	otherwise:	$r_v = -1$
neighbours	if $out_v = out_o$ and $0 < \|N(v)\| < 3$:	$r_v = 1$
	elif $out_v = out_o$ and $\|N(v)\| \geq 3$:	$r_v = 0.5$
	elif $out_v = out_o$ and $\|N(v)\| = 0$:	$r_v = 0.1$
	elif $0 < \|N(v)\| < 3$:	$r_v = -0.2$
	elif $\|N(v)\| \geq 3$:	$r_v = -0.5$
	otherwise:	$r_v = -1$

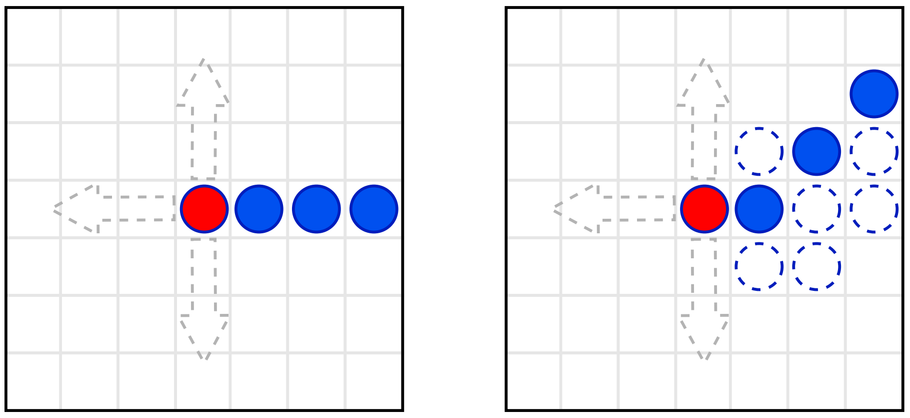
Table 4.4: Description of the reward functions used in MARL scenarios

connected graph. In the case of Transmission, this was necessary to solve the task, while in Moving it helped to learn the communication part, as communication only happens when the agents are connected in the graph. The 'random' placement strategy randomly positions the workers in the environment, possibly forming disconnected graphs from the start. Both the direction of the initial placement and the subsequent shifts were randomly determined at the beginning of each episode. Examples placements are shown in Figure 4.5.

The first stage of the curriculum used multidirectional placement. Unidirectional placement led to overfitting of the models to transmit information only from left to right and was therefore excluded from the curriculum. After solving this level, the placement strategy changed to random multidirectional. The random shifts introduced new graph topologies into the learning process, while the model had already learned to transmit information in all directions.

In the last curriculum level, episodes got longer and the output of the oracle changed with a probability of 10% after a cooldown period, we call this 'oracle switching'. The cooldown period after each oracle switch gives the agents enough time to, at least theoretically, propagate the new oracle output to everyone.

For the moving tasks, we also increased the size of the grid as the curricula progressed. We started small to make the agents reach connected states more



(a) Example of a multidirectional placement.

(b) Example of a randomised multidirectional placement.

Figure 4.5: Examples of placements to the right; dotted arrows indicate potential placement directions, dotted circles indicate potential worker locations.

often at the beginning of the learning process, when movement is random. However, it turned out that it wasn't necessary to gradually increase the grid size, as the policies learned on small grids were almost identical to the ones on large ones.

We considered a stage solved if the minimum achieved reward out of 30 episodes was higher than half of the maximum achievable reward.

All levels except the last, where oracle switching occurs, were trained with short episodes of length $T = 16$. The purpose of the fast reset is to avoid wasting training time if the agents diverge into bad states at the beginning of the learning process. This is especially true when movement is involved and the agents disconnect quickly due to random movements. For the last level we increased the episode length to $T = 64$ to allow multiple oracle switches.

4.7 Visualisation

When referring to the output of the agents in the Transmission, Transmission Extended, Moving and Moving History tasks, we will use colours to describe the numerical output. This makes it easier to visualise the results.

CHAPTER 5

Experiments

5.1 Hyperparameters

For each task, we tuned the hyperparameters by training 200-300 different trials. The algorithm parameters were sampled from the ranges described in table 5.1, while the model parameters were sampled from the ranges described in table 5.2. The results reported were generated by the best performing model across all trials. Note that the 'Message passing rounds' parameter is only sampled for the critic network, unless otherwise stated, the actor has $K = 1$.

Parameter	**Sample range**
Batch size	500
Learning rate	$[0.00003, 0.003]$
Discount factor γ	0.99
PPO clip parameter ϵ	$\{0.1, 0.2\}$
PPO value loss coefficient c_1	$[0.5, 1]$
PPO entropy coefficient c_2	$\{0, 0.1, 0.01\}$
GAE λ	$[0.9, 1]$
KL initial coefficient	$\{0.9, 1\}$
KL target value	$[0.003, 0.03]$
Global gradient norm clipping	1

Table 5.1: Algorithm hyperparameter sampling ranges.

Parameter	**Sample range**
Encoding size m	$\{8, 16, 32\}$
Message passing rounds K	$\{1, .., 8\}$
MLP number hidden layers	$\{0, 1, 2\}$
MLP hidden layer size	$\{8, 16, 32, 64\}$

Table 5.2: Network hyperparameter sampling ranges. MLP parameters only apply for the GINE convolution network $\mathbf{\Theta}_{MLP}$.

5.2 Lever Pulling

We conducted experiments with all three convolutions. As a baseline, we use random pulling, which gives a theoretical average reward of 0.67. The results are in the table 5.3.

Model	**Convolution**	**Rounds**	**Result**
Random	-	-	0.67
CommNet	-	2	**0.94**
Ours	Transformer	2	0.84 ± 0.12
Ours	Transformer	1	0.82 ± 0.13
Ours	GATv2	1	0.67 ± 0.15
Ours	GINE	1	0.66 ± 0.14

Table 5.3: MARL Lever Pulling results: Shows mean and standard deviation of the number of distinct levers pulled divided by the total levers ('Result'), along with convolution type ('Convolution') and message passing rounds ('Rounds'). Results were obtained from 250 independent episodes, each lasting $T = 1$.

Only the Transformer convolution was able to successfully complete the task better than random pulling. To have a comparable setup to CommNet, we additionally trained a model with the Transformer convolution that used two rounds of message passing. While showing that communication with our setup is possible, CommNet still outperformed our best model by a wide margin.

5.3 Transmission

We conducted experiments for both the Transmission and Transmission Extended tasks, using all three convolutions and three different reward functions. In addition, we studied the performance in both the SARL and MARL scenarios. Workers were placed according to the randomised multidirectional placement strategy, coupled with oracle switches set to a probability of 10% after the cooldown period. To maintain consistency, the episode length was adjusted to maintain an average of 8 oracle switches per episode, taking into account the extension of the cooldown period with longer communication lines as the number of workers increases.

n	**Grid size**	T
4	15x15	100
20	41x41	200
40	83x83	250

Table 5.4: Grid size and episode length ('T') in relation to the number of workers ('n') in the evaluation setup.

Table 5.5 shows that when trained with the shared binary reward, none of the convolutions perform better than random. We observerd the behaviour of the GATv2 policy to always output the colour blue. This results in a maximum

Reward	Convolution	Result
random	-	0.2 ± 0.05
shared binary	GINE	0.2 ± 0.24
	GATv2	**0.25 ± 0.23**
	Transformer	0.18 ± 0.19
shared sum	GINE	0.23 ± 0.13
	GATv2	**0.48 ± 0.29**
	Transformer	0.2 ± 0.09

Table 5.5: SARL Transmission results. Shows mean and standard deviation for the percentage of correct outputs ('Result') for models trained with the reward function ('Reward') and convolution type ('Convolution'). Both training and evaluation were conducted with $n = 4$ workers. Results were obtained from 50 independent episodes, for grid size and episode length refer to table 5.4.

reward only if the oracle also outputs blue, but deteriorates to 0 if any other colour is the output. Figure 5.1 visualizes this policy.

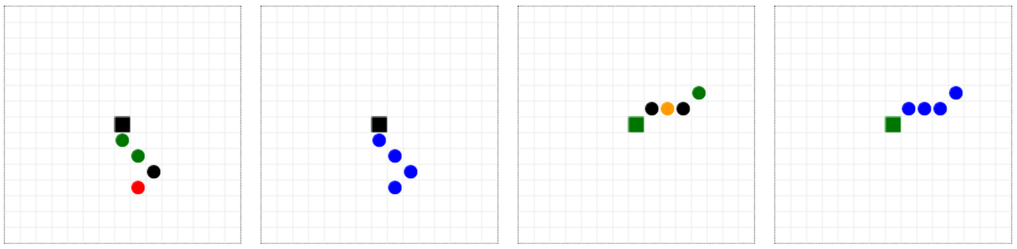
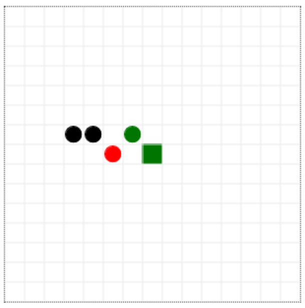
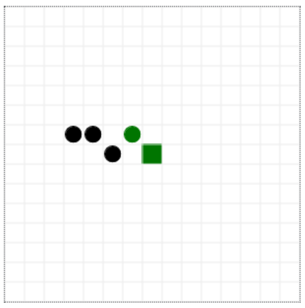

(a) t=0 (b) t=1 (c) t=0 (d) t=1

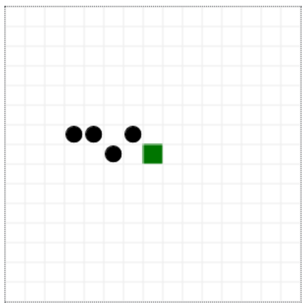
Figure 5.1: Visualization of the GATv2 policy trained with a shared binary reward, showing two sample trajectories labeled as (a), (b) and (c), (d) at time t. In both trajectories, all workers consistently converge to outputting blue, regardless of the oracle's output.

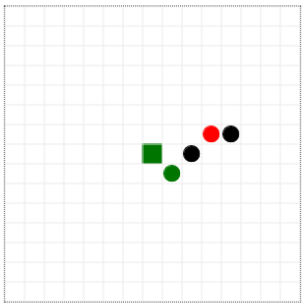
Training with the shared sum reward gives a slightly more fine grained feedback, not only signaling solution correctness but also quantifying the degree of deviation from correctness. The GATv2 policy overcame it's local optimum, but it has not yet found an optimal policy. It shows proficiency in propagating the oracles output from left to right, but struggles doing so in the opposite direction, which explains the average reward of 0.5. When the communication line is oriented vertically, it may lead to either outcome. Figure 5.2 illustrates an example of this behaviour.

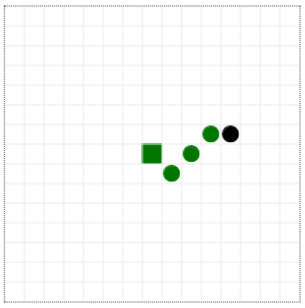
Training the same task in the MARL scenario improved the performance of all convolutions. However, only with the Transformer convolution a close-to-optimal policy was learned. We will therefore focus our discussion on the results achieved by the transformer convolution. Table 5.6 displays the results. The


(a) t=0


(b) t=1


(c) t=2


(d) t=0


(e) t=1

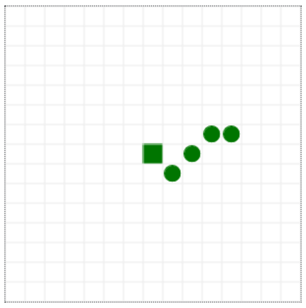
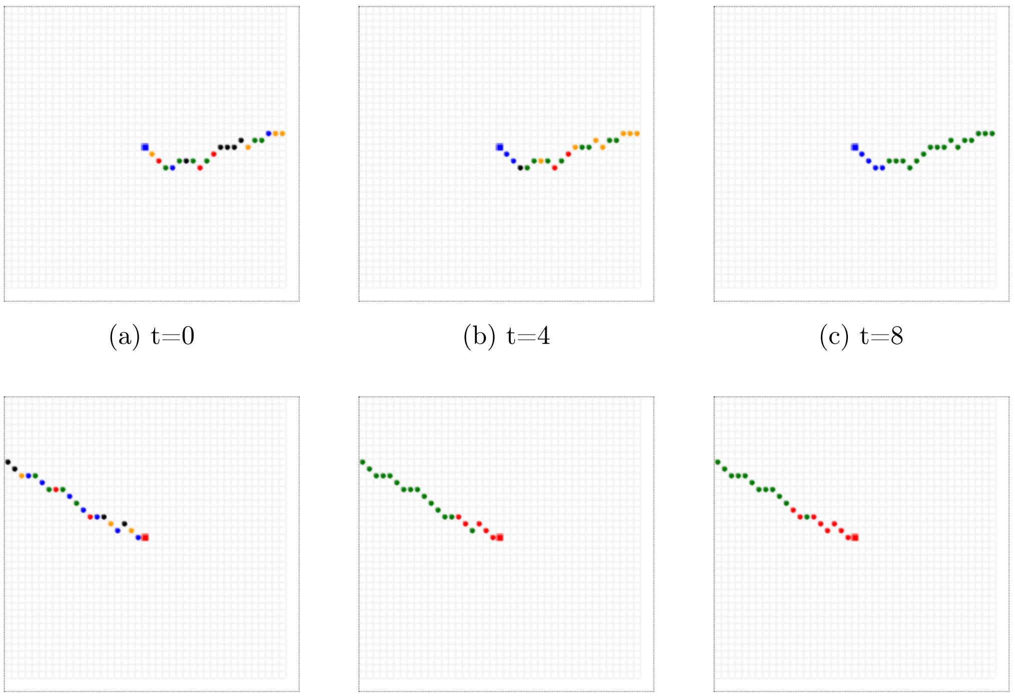

(f) t=2

Figure 5.2: Visualization of the GATv2 policy trained with a shared sum reward, showing two sample trajectories labeled as (a), (b), (c) and (d), (e), (f) at time t. The workers in the first trajectory converge towards the colour black, which corresponds to the leftmost worker at t=0. In the second trajectory, successful convergence to the colour green is achieved, while the oracle represents the leftmost agent in this configuration.

policy trained in the MARL scenario with Transformer convolution using shared binary reward shows an increase in performance compared to the SARL GATv2 policy result. Upon inspection of the behaviour, the policy successfully converges in all directions if the oracle outputs yellow or blue, but not for the other colors. This finding explains the average performance of $0.4 = 2/5$.

The performance of the policy trained with shared sum is comparable to that of the SARL GATv2 model. Upon inspection of the behaviour, it was found that it successfully converges if the output of the oracle is yellow, black or blue. However, the transmission speed is not optimal. For instance, in the case of n=4, the optimal time to convergence is 2, which explains the slightly lower average performance of 0.5, which is less than $3/5$.

Giving individual rewards to workers based on their output enabled the task to be solved consistently when evaluated with $n = 4$ workers. The same policy was used to evaluate performance as the communication line grew, as seen in the results of $n = 20$ and $n = 40$. It was observed that workers closer to the oracle converged to the correct colour, while those farther away were likely to output green at some point. Around $t = 8$, a two colour pattern usually established. Afterward, the boundary between the two colours is alternately pushed back and forth, sometimes resulting in one colour completely overtaking the other. This pattern is visualised in Figure 5.3.



(a) t=0 (b) t=4 (c) t=8

(d) t=0 (e) t=8 (f) t=12

Figure 5.3: Visualization of the Transformer policy trained with individual reward, showing two sample trajectories labeled as (a), (b), (c) and (d), (e), (f) at time t with $n = 20$. The workers converge to a two-colour pattern, with those close to the oracle converging to the correct colour, while those farther away were more likely to output green at some point. Subfigures (e) and (f) illustrate the movement of the boundary between the colours.

Adjacent workers consistently propagate the output of the oracle in all directions. However, middle workers can only determine the origin of the oracle's output based on the messages received from neighbouring workers. Therefore, they must establish a certain level of confidence to decide from which direction to integrate information. To investigate whether knowledge of their position in relation to the oracle assists in determining the direction of integration, we expanded the worker's state in the Transmission Extended task.

The performance became more stable with longer communication lines, looking at the results for $n = 20$ and $n = 40$. After some time, the same two-color pattern typically emerges. One significant difference in behaviour, compared to the policy trained without considering the oracle's relative position, is that workers far from the oracle still converge to the same colour. However, this colour no longer defaults to green or any single colour. We evaluated both tasks with identical setups but without oracle switching for $T = 30$, focusing solely on convergence behaviour. The results are shown in Figure 5.4.

Reward	**# Agents**		
	$n = 4$	$n = 20$	$n = 40$
shared binary	0.39 ± 0.26	0.24 ± 0.18	-
shared sum	0.5 ± 0.22	0.26 ± 0.13	-
shared sum (SARL GATv2)	0.48 ± 0.29	0.26 ± 0.19	0.24 ± 0.16
individual	$\mathbf{0.91 \pm 0.05}$	0.35 ± 0.13	0.25 ± 0.14
individual$^+$	0.89 ± 0.08	$\mathbf{0.4 \pm 0.17}$	$\mathbf{0.38 \pm 0.19}$

Table 5.6: MARL Transmission results of the Transformer policy. Shows mean and standard deviation for the percentage of correct outputs for models trained with the reward function ('Reward'), evaluated with n workers. Results for reward 'individual$^+$' were generated with the Transmission Extended task setup. All models were trained with $n = 4$ workers. Results were obtained from 50 independent episodes in cases $n \in \{4, 20\}$ and 25 independent episodes in the case $n = 40$. For grid size and episode length refer to table 5.4.

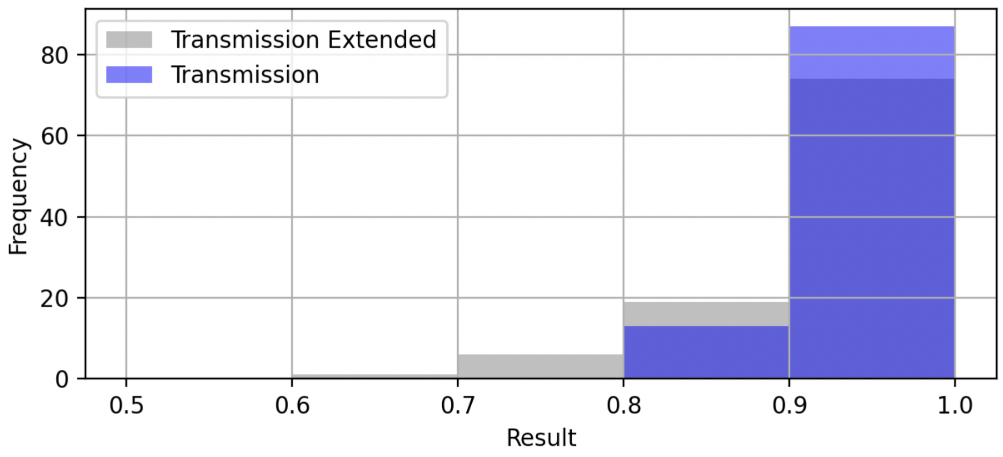


Figure 5.4: Histogram illustrating the fraction of correct outputs at t=30 from 100 independent trials for both the Transmission and Transmission Extended tasks.

5.4 Moving

Experiments were conducted for both the Moving and Moving History tasks using three distinct reward functions. The experiments were conducted exclusively within the MARL scenario utilizing the Transformer convolution due to the satisfactory performance observed in prior tasks. Workers were positioned using a randomized multidirectional placement strategy, and oracle switches were set to occur at a 10% probability after the cooldown period. We additionally evaluated the random placement strategy, to compare the dependence on the initial graph structure. To ensure consistency, we again adjusted the length of the episode according to the values specified in Table 5.4.

The objective of the spread reward function is to incentivise workers to create a communication graph that covers a lot of area. However, none of the trained models were able to surpass the local optimum of converging to a single colour and distancing themselves as far as possible from the oracle. This situation results in a maximum score only if the oracle also outputs this colour, but deteriorates to 0 if any other number is chosen. Figure 5.5 illustrates this phenomenon.

To counteract this strategy, we adjusted the reward to incentivise workers to maintain communication with the oracle. This reward promotes connectivity. Despite this adjustment, workers still tend to gravitate towards the edges of the environment, where they can potentially maximise the reward. Although the learned policy succeeds in increasing the reward by remaining near the oracle for a longer period, workers struggle to sustain this proximity. The connection to the oracle is eventually lost, causing the workers to drift away from it once again, as shown in Figure 5.6.

Once the connection between the last worker and the oracle was lost, the workers were unable to reconnect to the graph. The graph deteriorated, which affected the information the workers' could use to compute actions, as illustrated in Figure 5.7. A potential improvement was to include a history of the relative distances of an edge in the edge state. However, a drawback of this approach is that if an edge disappears, meaning two agents become disconnected, the history is lost. We decided on another approach, which was to enhance the state by including the worker's position relative to the oracle. This resulted in the task Moving History.

The presented heatmap in Figure 5.9 shows that access to absolute positions improves resilience against graph deterioration. Workers remain close to the oracle and maintain connectivity. With random initial placements, the workers show a slight drift towards the oracle. However, there are still some workers that remain lost at the borders. As connectivity increases, there is a slight improvement in performance, as shown in Table 5.8. Additionally, increasing the size of the workers' hidden state to 64 results in further performance enhancement. The heatmap in Figure 5.10 shows a movement pattern with an even more pronounced drift towards the center, even when workers are randomly placed.

For the final experiment, we used the neighbours reward function to train the policies. Unlike the spread connected reward, the neighbours reward is based on the agent's local neighborhood topology rather than its distance to the oracle. This encourages agents to cover a larger area based on the number of neighbors rather than their proximity to the oracle. The main policy that emerged was for all agents to move as close to the oracle as possible, as shown in Figures 5.11 and 5.12. They then replicate the oracle's output consistently from their direct edge to it. However, in the Moving task, this policy encounters the same issue of disconnectivity, which is mitigated by incorporating historical positional information.

Reward	**Placement**	**# Agents**	
		$n = 4$	$n = 20$
spread	multi directional	0.21 ± 0.22	0.21 ± 0.18
	random	0.16 ± 0.2	-
spread (connected)	multi directional	0.4 ± 0.23	0.25 ± 0.17
	random	0.2 ± 0.15	-
neighbours	multi directional	**0.71 ± 0.19**	**0.32 ± 0.18**
	random	0.34 ± 0.18	0.26 ± 0.11

Table 5.7: MARL Moving Task Results. Shows mean and standard deviation for the percentage of correct outputs for models trained with the reward function ('Reward'), evaluated with n workers with initial placement strategy ('Placement'). All models were trained with $n = 4$ workers and the Transformer convolution. Results were obtained from 50 independent episodes in cases $n \in \{4, 20\}$ and 25 independent episodes in the case n = 40. For grid size and episode length refer to table 5.4.

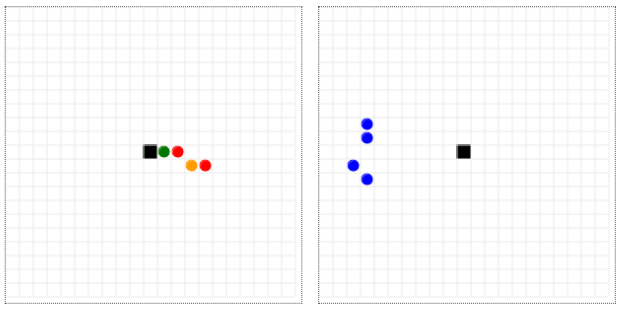

(a) t=0 (b) t=6

Figure 5.5: Visualization of the Transformer policy trained with the spread reward, showing a sample trajectory at two points in time. The workers consistently converge to blue and moves as far away from the oracle as possible.

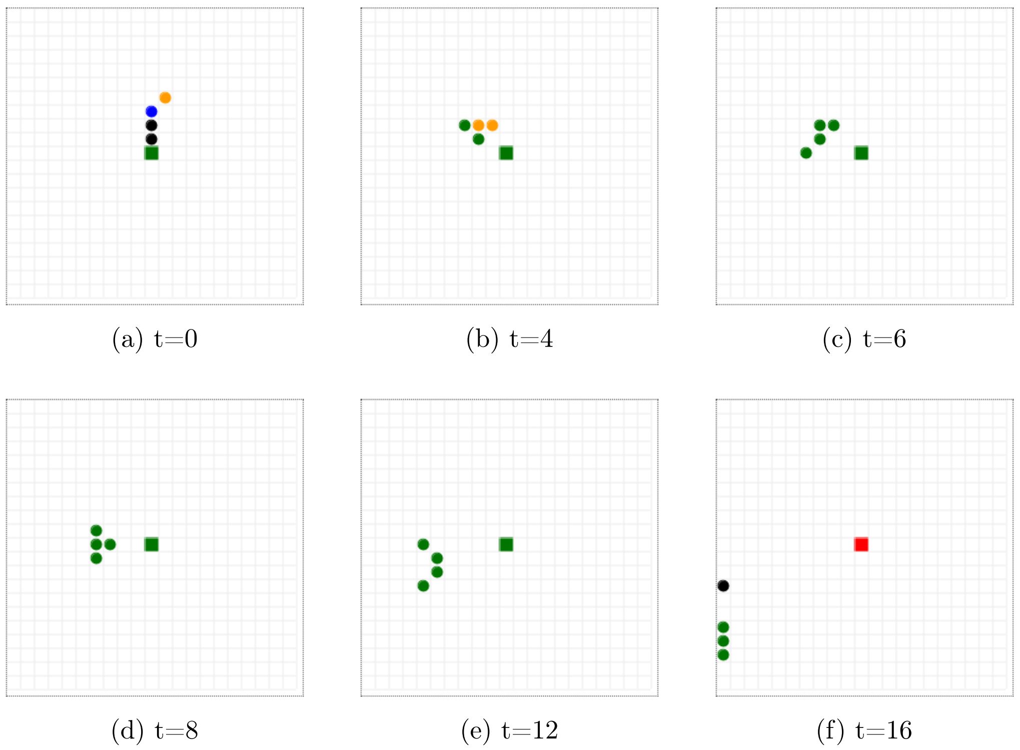


Figure 5.6: Visualization of the Transformer policy trained with the spread connected reward, showing a sample trajectory at six points in time. The workers successfully copy the output of the oracle by remaining connected to it. Once the connection is lost, the workers revert to trying to maximise the reward by being as far away from the oracle as possible.

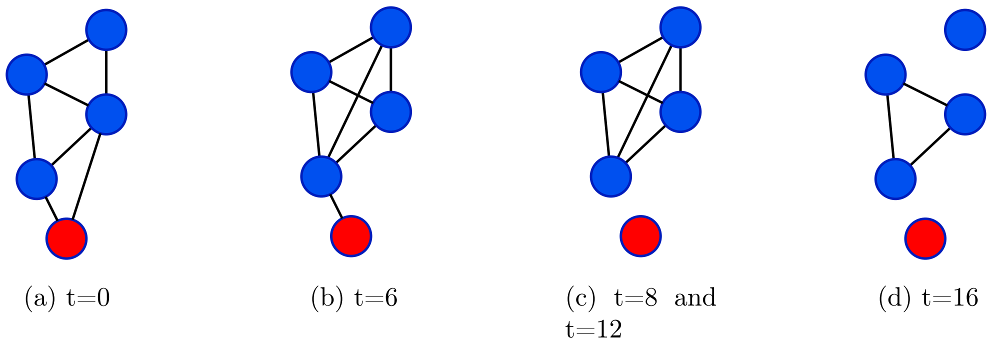


Figure 5.7: Visualization of the detoriation of the computation graph from Figure 5.6. After $t \geq 8$, the agents are unable to determine the location of the oracle unless it was encoded in their hidden vector.

Reward	**Placement**	**# Agents**		
		$n = 4$	$n = 20$	$n = 40$
spread	multi directional	0.19 ± 0.17	0.23 ± 0.19	-
	random	0.22 ± 0.14	0.21 ± 0.09	-
spread (connected)	multi directional	0.47 ± 0.17	0.39 ± 0.15	0.38 ± 0.18
	random	0.42 ± 0.16	0.37 ± 0.1	0.38 ± 0.21
spread (connected) *large*	multi directional	0.66 ± 0.08	0.55 ± 0.15	0.42 ± 0.18
	random	0.54 ± 0.15	0.55 ± 0.12	0.45 ± 0.13
neighbours	multi directional	**0.91 ± 0.11**	**0.82 ± 0.18**	**0.8 ± 0.16**
	random	0.80 ± 0.18	0.78 ± 0.19	0.78 ± 0.20

Table 5.8: MARL Moving History results. Shows mean and standard deviation for the percentage of correct outputs for models trained with the reward function ('Reward'), evaluated with n workers with initial placement strategy ('Placement'). All models were trained with $n = 4$ workers and the Transformer convolution. Results were obtained from 50 independent episodes in cases $n \in \{4, 20\}$ and 25 independent episodes in the case $n = 40$. For grid size and episode length refer to table 5.4.

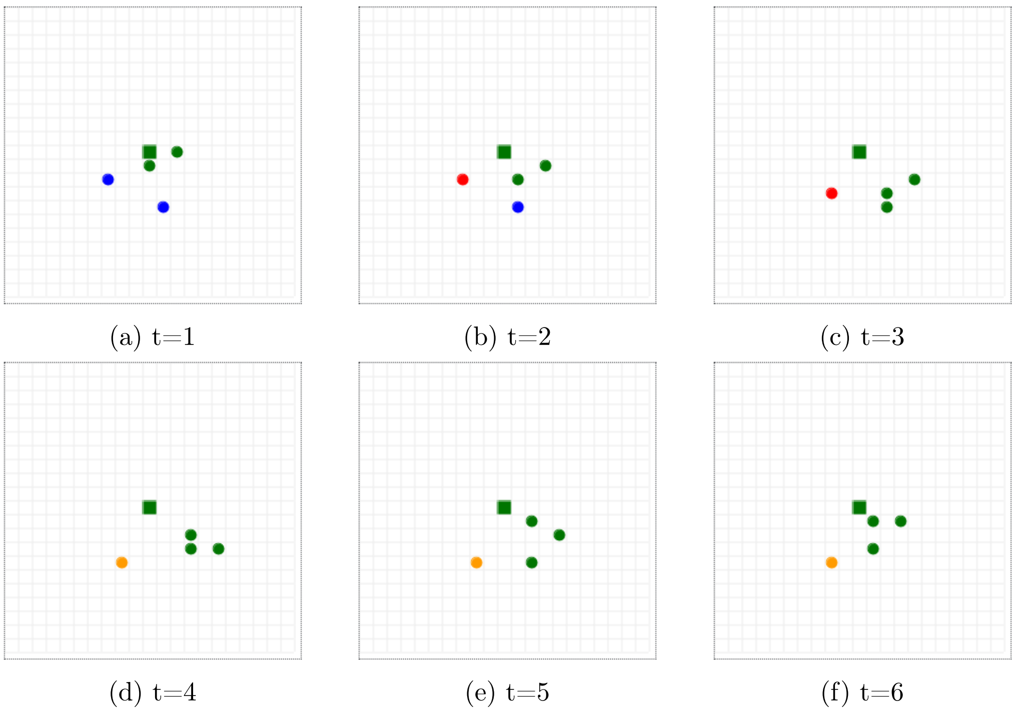

(a) t=1 (b) t=2 (c) t=3
(d) t=4 (e) t=5 (f) t=6

Figure 5.8: Visualization of the Transformer policy trained with the spread connected reward on the Moving History task, showing a sample trajectory at six points in time. The workers successfully reconnect to the oracle after being completely disconnected at $t = 3$ and $t = 4$.

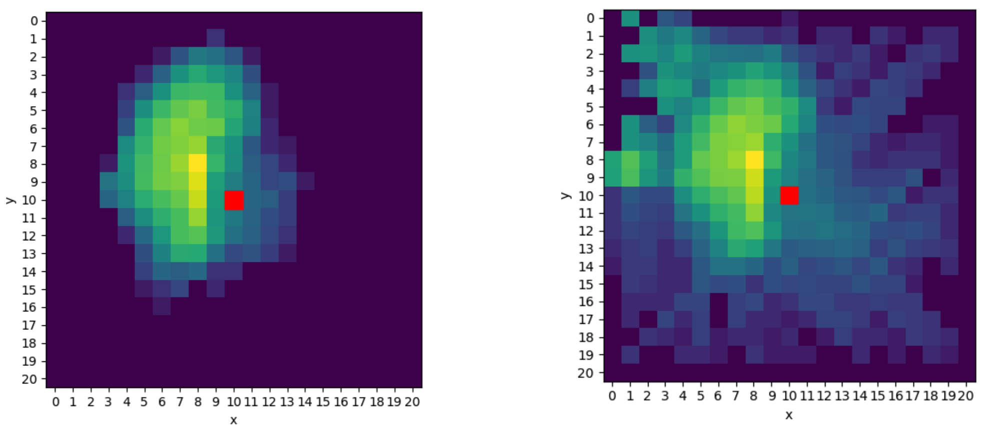


Figure 5.9: Heatmap of the workers locations on a logarithmic scale with the oracle position as red square. The heatmap was generated over 100 independend trajectories produced by the policy trained with the spread connected reward function, with $n = 4$, and placement strategy multi directional (left) and random (right), respectively.

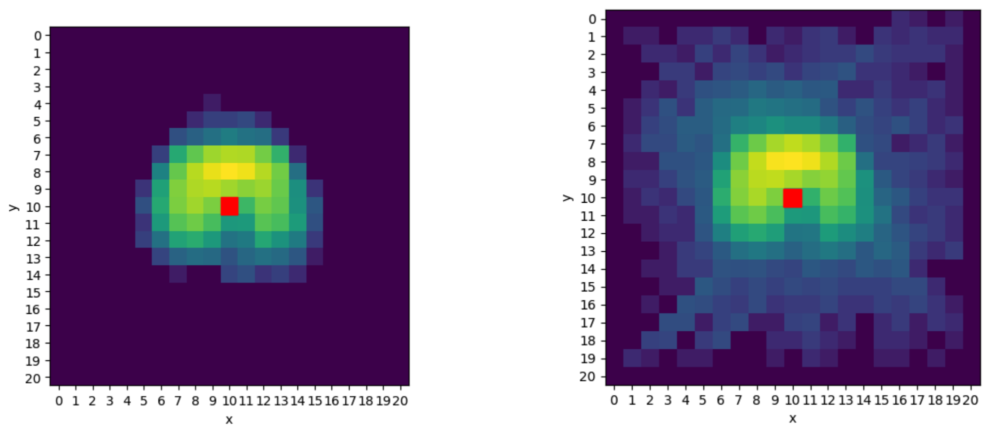


Figure 5.10: Heatmap of the workers locations on a logarithmic scale with the oracle position as red square. The heatmap was generated over 100 independend trajectories produced by the policy trained with the spread connected reward function, with $n = 4$, $m = 64$, and placement strategy multi directional (left) and random (right), respectively.

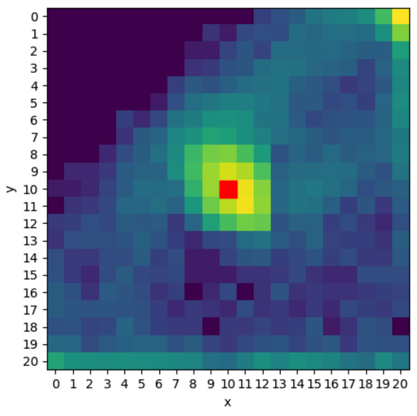
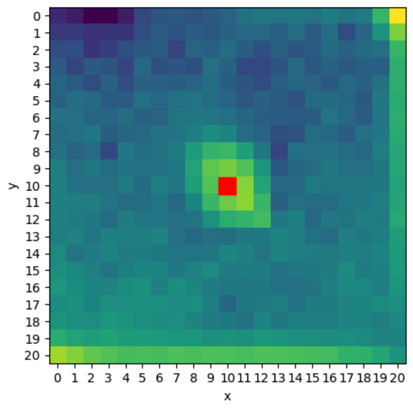



Figure 5.11: Heatmap of the workers locations on a logarithmic scale with the oracle position as red square. The heatmap was generated over 100 independend trajectories produced by the policy trained with the neighbourhood reward function on the Moving task, with $n = 4$, and placement strategy multi directional (left) and random (right), respectively.

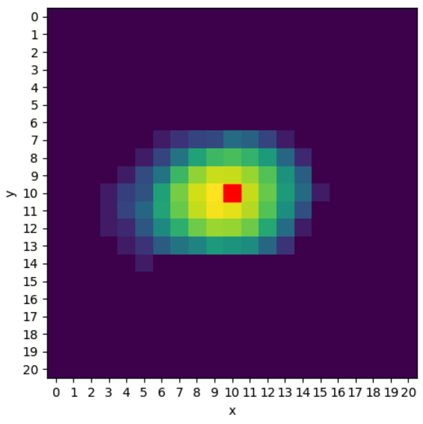
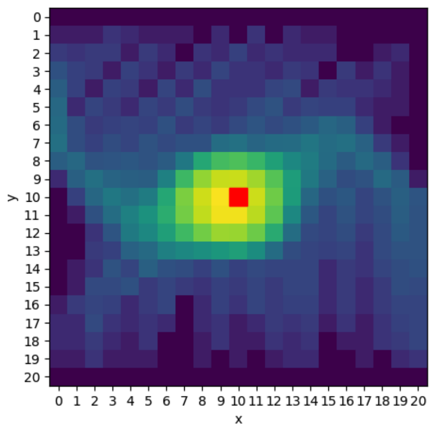



Figure 5.12: Heatmap of the workers locations on a logarithmic scale with the oracle position as red square. The heatmap was generated over 100 independend trajectories produced by the policy trained with the neighbourhood reward function on the Moving History task, with $n = 4$, and placement strategy multi directional (left) and random (right), respectively.

5.5 MPE Spread

Experiments were conducted with models using the Transformer convolution. We replicated the training setup presented in [23] to make our results comparable. This includes time discretisation of $dt = 0.1$ with a total episode length of $T = 25$. Also, for the experiments with $n = 50$, the size of workers were scaled down by a factor of 25. It is to note that this makes collisions more unlikely. The parameterisation of the MPE environment was not included in their paper, we assumed that they used the default parameters of the official implementation.

Model	**Rounds**	**# Agents**	
		$n = 3$	$n = 50$
Random	-	0.75 ± 0.24	0.18 ± 0.03
MAADPG	-	0.22 ± 0.010	-
MAAC	-	0.19 ± 0.012	-
ATOC	-	0.17 ± 0.007	0.09
HAMA	-	$\mathbf{0.15 \pm 0.008}$	**0.07**
Ours	1	0.46 ± 0.34	0.16 ± 0.04
Ours	8	0.25 ± 0.22	0.23 ± 0.11
Ours$^+$	8	0.27 ± 0.22	0.16 ± 0.03

Table 5.9: MARL MPE Spread result. Shows mean and standard deviation for the mean negative reward, whereas evaluated with n workers. All models were trained with $n = 3$ workers, while our model utilized the Transformer convolution. The results were obtained from 50 independent episodes.

The models successfully learn a policy to occupy all landmarks, as illustrated in Figure 5.13. When landmarks are spread apart, each one is occupied by a single agent. However, when the landmarks are close together, agents may compete for the same landmark, resulting in collisions, as depicted in Figure 5.14. These collisions have a significant negative impact on performance. We observed that increasing the number of message passing rounds for the actor improves performance. Furthermore, introducing a hidden vector alongside the increased message passing rounds alters the workers' strategy to be less aggressive, resulting in fewer collisions but greater distance from the landmarks. This strategy proves to be more effective as the number of agents increases. Despite these improvements, the reference models still outperform our models by a significant margin.

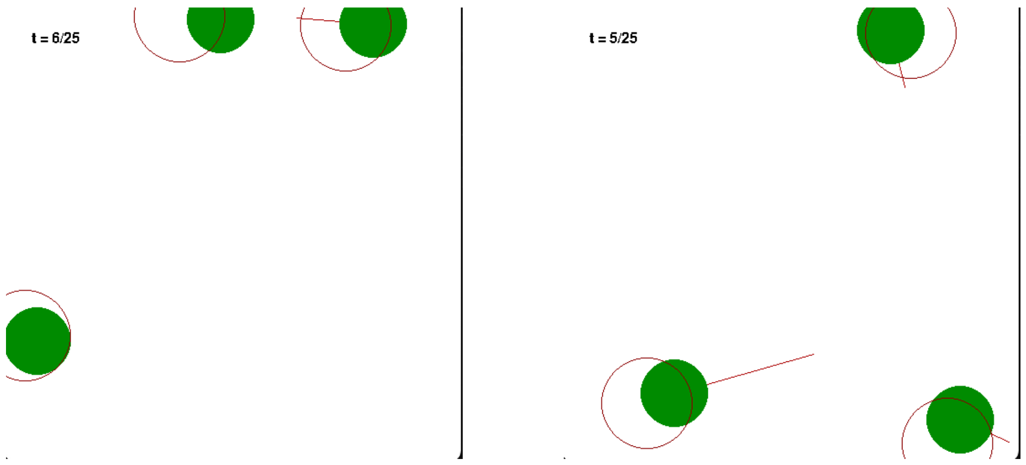
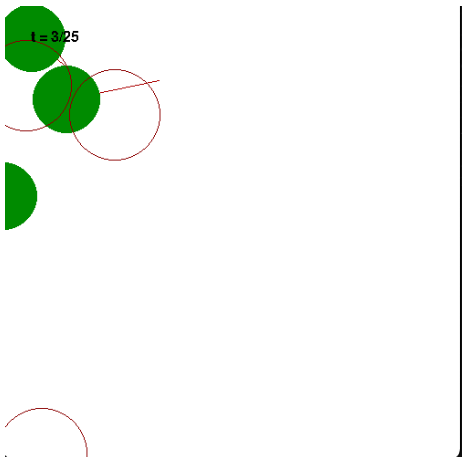
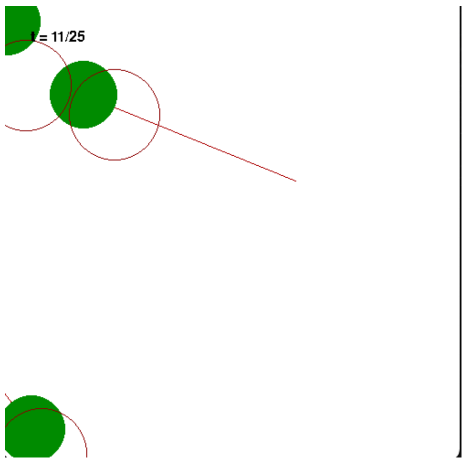


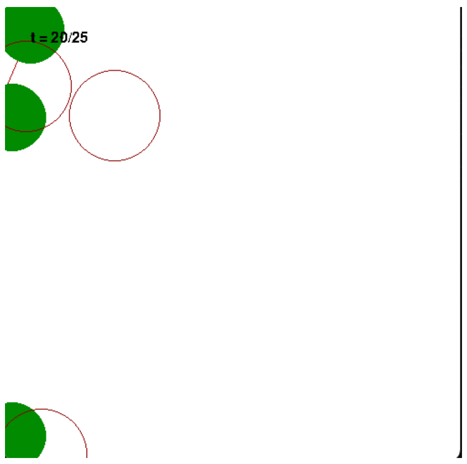
Figure 5.13: Visualization of the MPE Spread task successfully solved by the Transformer policy if landmarks are far appart.



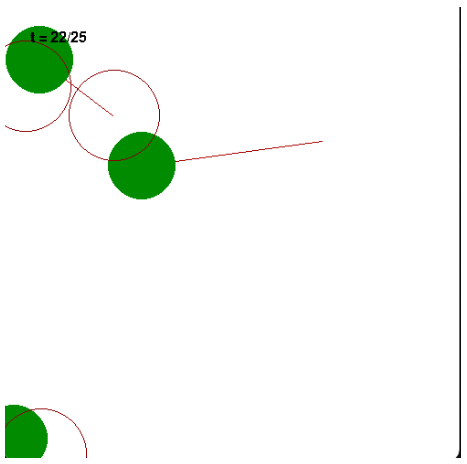
(a) t=3



(b) t=11



(c) t=20



(d) t=22

Figure 5.14: Visualisation of the MPE Spread task successfully solved by the Transformer policy when landmarks are close together. Agents spread close to each other, as shown in figure (a). One agent manages to grab the unoccupied landmark, while the other two fight for one.

Chapter 6

Conclusion

In this work, we have developed an architecture suitable for handling multi-agent environments with a large number of agents. Through the utilization of graph convolutions, policies can be trained efficiently to navigate complex scenarios. Leveraging tools such as Ray Tune and structured observation spaces, we have demonstrated the flexibility to adapt graph convolutions used for actor and critic networks, as well as the graph and edge states of the graphs. This adaptability enables trained policies to be seamlessly applied to scenarios with varying numbers of agents, showcasing the versatility and scalability of our approach.

The results of the Transmission task compare learning policies with the SARL scenario against policies learned with the MARL scenario. Based on the results of tables 5.6 and 5.5, we conclude that for the tasks we have chosen, the feedback given by shared rewards is too sparse to learn effectively. In the case of shared binary rewards, the probability that all agents output the same colour at the beginning of the learning process is very low. Reducing the granularity of the reward feedback by using shared sum or individual rewards proved to be beneficial for the learning process.

The comparison between different convolutional methods for policy implementation highlighted the effectiveness of incorporating attention mechanisms. In particular, attention to agents along the communication axis proved crucial for solving the Transmission task, as evidenced by the performance of the Transformer policies. However, results from the Moving task, particularly when comparing policies trained with the spread connected reward to those trained with the neighbourhood reward, indicate the challenge agents face in deciding which information to prioritise. The performance of the neighbourhood reward trained policy suggests a greater reliance on information from the oracle compared to other sources. In scenarios with long communication lines or widely dispersed communication graphs resulting from spread connected reward, agents face greater difficulty in selecting the correct information and generating the correct output.

Including the absolute position in the worker states proved beneficial for information propagation, as shown by the results in Table 5.6, which compares rewards on an individual basis. However, as shown in the histogram in Figure

5.8, the policy trained with absolute positions did not converge on agent states as effectively as the policy trained without them. Nevertheless, the lack of convergence to a default colour for agents far from the oracle, coupled with the improved performance over longer communication links, suggests that knowledge of the absolute state contributed to the development of a more adaptive policy. In addition, the inclusion of absolute position showed that workers learned to maintain connectivity, while a history of position allowed brief deviations from connected parts, as observed in the heatmaps in Figures 5.9, 5.12, 5.11 and 5.10. We conclude that the convolutions we used are not strong enough to encode enough positional information in the hidden vector to solve the tasks with a similar performance but without absolute position.

The performance results of the MPE spread task, as displayed in Table 5.9, underscore the potential of our approach to address more complex challenges beyond grid world abstractions. However, the findings also highlight the necessity for further research and development to fully capitalize on this potential.

Our most pressing improvement recommendation is to incorporate convolutions that can more effectively leverage edge attributes. In addition, further research is needed to quantify the potential improvement that additional rounds of message passing can bring. While we have observed some evidence of its effectiveness and potential, empirical evidence is needed to numerically validate these findings.

Bibliography

[1] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, “Mastering chess and shogi by self-play with a general reinforcement learning algorithm,” *arXiv preprint arXiv:1712.01815*, 2017.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, “Grandmaster level in starcraft ii using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

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

[5] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, “Emergent tool use from multi-agent autocurricula,” *arXiv preprint arXiv:1909.07528*, 2019.

[6] M. Gori, G. Monfardini, and F. Scarselli, “A new model for learning in graph domains,” in *Proceedings. 2005 IEEE international joint conference on neural networks, 2005.*, vol. 2, pp. 729–734, IEEE, 2005.

[7] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[8] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, “Rllib: Abstractions for distributed reinforcement learning,” in *International conference on machine learning*, pp. 3053–3062, PMLR, 2018.

[9] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, “Tune: A research platform for distributed model selection and training,” 2018.

[10] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," 2019.

[11] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, vol. 55, no. 2, pp. 895–943, 2022.

[12] S. Sukhbaatar, R. Fergus, *et al.*, "Learning multiagent communication with backpropagation," *Advances in neural information processing systems*, vol. 29, 2016.

[13] Y. Hoshen, "Vain: Attentional multi-agent predictive modeling," *Advances in neural information processing systems*, vol. 30, 2017.

[14] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, *et al.*, "Interaction networks for learning about objects, relations and physics," *Advances in neural information processing systems*, vol. 29, 2016.

[15] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *International conference on machine learning*, pp. 2961–2970, PMLR, 2019.

[16] J. Jiang and Z. Lu, "Learning attentional communication for multi-agent cooperation," *Advances in neural information processing systems*, vol. 31, 2018.

[17] D. Wang, J. Lin, P. Cui, Q. Jia, Z. Wang, Y. Fang, Q. Yu, J. Zhou, S. Yang, and Y. Qi, "A semi-supervised graph attentive network for financial fraud detection," in *2019 IEEE International Conference on Data Mining (ICDM)*, pp. 598–607, IEEE, 2019.

[18] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," *arXiv preprint arXiv:1709.04875*, 2017.

[19] H. Chereda, A. Bleckmann, K. Menck, J. Perera-Bel, P. Stegmaier, F. Auer, F. Kramer, A. Leha, and T. Beißbarth, "Explaining decisions of graph convolutional neural networks: patient-specific molecular subnetworks responsible for metastasis prediction in breast cancer," *Genome medicine*, vol. 13, pp. 1–16, 2021.

[20] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[21] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[22] Z. Zhang, P. Cui, and W. Zhu, “Deep learning on graphs: A survey,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 249–270, 2020.

[23] H. Ryu, H. Shin, and J. Park, “Multi-agent actor-critic with hierarchical graph attention network,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 7236–7243, 2020.

[24] J. Sheng, X. Wang, B. Jin, J. Yan, W. Li, T.-H. Chang, J. Wang, and H. Zha, “Learning structured communication for multi-agent reinforcement learning,” *Autonomous Agents and Multi-Agent Systems*, vol. 36, no. 2, p. 50, 2022.

[25] M. Hüttenrauch, A. Šošić, and G. Neumann, “Local communication protocols for learning complex swarm behaviors with deep reinforcement learning,” in *Swarm Intelligence: 11th International Conference, ANTS 2018, Rome, Italy, October 29–31, 2018, Proceedings 11*, pp. 71–83, Springer, 2018.

[26] I. Mordatch and P. Abbeel, “Emergence of grounded compositional language in multi-agent populations,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[27] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” *Advances in neural information processing systems*, vol. 30, 2017.

[28] F. A. Oliehoek, “Decentralized pomdps,” in *Reinforcement learning: state-of-the-art*, pp. 471–503, Springer, 2012.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[30] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[31] S. M. Kakade, “A natural policy gradient,” *Advances in neural information processing systems*, vol. 14, 2001.

[32] S. Levine, “Policy gradients,” 2007. Available at `http://rail.eecs.berkeley.edu/deeprlcourse-fa17/index.html`, Accessed: 2024-02-02.

[33] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, “The surprising effectiveness of ppo in cooperative multi-agent games,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 24611–24624, 2022.

[34] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” 2017.

[35] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[36] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, PMLR, 2016.

[37] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[38] V. Elvira and L. Martino, "Advances in importance sampling," *arXiv preprint arXiv:2102.05407*, 2021.

[39] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*, pp. 1263–1272, PMLR, 2017.

[40] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," *arXiv preprint arXiv:1905.12265*, 2019.

[41] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?," *arXiv preprint arXiv:2105.14491*, 2021.

[42] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," *arXiv preprint arXiv:2009.03509*, 2020.

[43] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, M. Hardt, B. Recht, and A. Talwalkar, "Massively parallel hyperparameter tuning," 2018.

[44] A. Stooke and P. Abbeel, "rlpyt: A research code base for deep reinforcement learning in pytorch," *arXiv preprint arXiv:1909.01500*, 2019.

[45] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," 2017.

[46] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, *et al.*, "Stable baselines," 2018.

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

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

[49] D. Masad, J. L. Kazil, *et al.*, "Mesa: An agent-based modeling framework.," in *SciPy*, pp. 51–58, Citeseer, 2015.

[50] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente, *et al.*, "Pettingzoo: Gym for multi-agent reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15032–15043, 2021.