This thesis has been submitted in fulfilment of the requirements for a postgraduate degree (e.g. PhD, MPhil, DClinPsychol) at the University of Edinburgh. Please note the following terms and conditions of use:

- This work is protected by copyright and other intellectual property rights, which are retained by the thesis author, unless otherwise stated.
- A copy can be downloaded for personal non-commercial research or study, without prior permission or charge.
- This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author.
- The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author.
- When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.
Advances in Open Ad Hoc Teamwork and Teammate Generation

Muhammad Arrasy Rahman

Doctor of Philosophy
Artificial Intelligence Applications Institute
School of Informatics
University of Edinburgh
2023
Abstract

Many real-world problems require an agent that can adapt its policy to efficiently collaborate with different groups of teammates whose composition may change over time. Previous work to design agents with such adaptive capabilities has been explored in the field of ad hoc teamwork. Given a predefined set of teammates for training, prior methods for ad hoc teamwork focused on training an agent to collaborate within a closed team where teammates remain in the environment during interaction with the trained agent. In this thesis, we consider ad hoc teamwork in open teams where agents with different fixed policies can enter and leave the environment.

This thesis contributes to the Graph-based Policy Learning (GPL) approach for ad hoc teamwork in open teams, assuming full observability of the environment. GPL leverages graph neural networks (GNNs) to predict teammates’ actions and estimate their effects on the trained agent’s returns. These predictions are then utilised to compute the trained agent’s optimal action-value function when dealing with open teams. We empirically demonstrate GPL’s effectiveness for training agents in ad hoc teamwork with open teams by showing it achieves significantly higher returns than agent policies resulting from various deep reinforcement learning baselines. Further analysis also demonstrates that GPL’s success results from effectively learning the effects of teammates’ actions towards the trained agent.

We also contribute to an extension of GPL to environments under partial observability. GPL’s extension to partially observable environments is based on different methodologies to maintain belief estimates over the latent environment states and team composition. The belief estimates are inferred based on the trained agent’s sequence of observations and utilised to compute the learning agent’s optimal policy under partial observability. Empirical results demonstrate that this extension can learn efficient open ad hoc teamwork policies under partial observability. Further analysis demonstrates that this efficiency results from accurately predicting the latent teammate actions and environment state.

The final contribution of this thesis is a method for the automated discovery of diverse training teammate types. This method is the first step to prevent a trained agent from performing poorly against previously unseen teammates with significantly different behaviour from those encountered during training. Our approach assumes closed environments and is based on the idea that an optimal set of training teammates consists of agents that require different best-response policies for optimal collaboration. Training against teammates from this set enables the trained agent to learn a broader
range of behaviours necessary for efficient collaboration in ad hoc teamwork. We fi-
nally demonstrate that our teammate generation approach improves the robustness of
a learner’s performance in ad hoc teamwork compared to alternative methods.
Lay Summary

It is impossible in many applications to provide a machine with all the instructions required to handle any scenario. Likewise, giving a machine all the necessary instructions to collaborate with different decision-makers that exhibit diverse behaviours is impractical. Thus, an ideal machine should autonomously be capable of adapting its behaviour to collaborate with different decision-makers.

Ad hoc teamwork is a research area concerned with designing adaptive machines that can collaborate with a wide range of teammates. Despite its potential usefulness, real-world applications of ad hoc teamwork is impeded by the strict assumptions upheld by existing methods. This work proposes approaches that remove two limiting assumptions underlying current ad hoc teamwork methods.

Our first idea, Graph-based Policy Learning (GPL), addresses the assumption that a machine’s collection of teammates during an interaction remains the same. For many real-world problems, addressing this assumption is crucial because a machine’s set of teammates can change throughout an interaction. GPL formulates a solution that relaxes this assumption by relying on predictive models that can deal with inputs of varying sizes to decide the machine’s behaviour. In this work, we formulate variations to GPL that enable a machine to collaborate with a changing collection of teammates in scenarios where it can and cannot see the entire state of the environment.

The second limiting assumption addressed in this work pertains to access towards other decision-makers for training. Existing ad hoc teamwork methods use the designed machine’s experience interacting with these decision-makers as a basis to learn adaptive behaviours to deal with other decision-makers. However, providing such decision-makers that will lead towards robust machines that can effectively collaborate with a wide range of decision-makers is often impossible. In this work, we provide a method called Best-Response Diversity (BRDiv) to generate decision-makers that facilitate the emergence of robust machines from ad hoc teamwork. For robust machines to result from AHT training, we argue that ensuring each decision-maker requires a machine to behave differently for effective collaboration is crucial.
Acknowledgements

My sincere gratitude goes to my supervisor, Dr. Stefano Albrecht, who has provided me with helpful feedback and guidance since my PhD began. With his useful advice and prior work in my research area, it is possible for all of the methods proposed in this work to come to fruition. Stefano has also provided me with essential knowledge which will be helpful for any endeavours that I will undertake in the future. I cannot thank him enough for all the valuable knowledge and experience I have gained from my interactions with him.

I also want to thank all my colleagues who gave me their valuable time to provide constructive advice and criticism to my work. Special thanks go towards Dr. Ignacio Carlucho, Niklas Höpner, and Elliot Fosong with whom I have regular discussions about ad hoc teamwork. For the countless times I felt stuck with my research, they were always there, ready to provide a spark of inspiration that eventually helped me move forward.

My most immense gratitude goes towards my parents, grandparents, and brother, who have supported me since the beginning. All that I have achieved today is only possible because of their long-lasting support. Even when we did not have much, they always found a way to provide me with support and the best education they could provide. No words are enough to express my gratitude for what they gave me.

Finally, I thank Farraz Theda, Fallon Chandra, and Ibrahim Ahmed for being such great friends during this journey. Even during the tough times in my PhD, my conversations with them helped keep me happy and motivated. One day, I hope to repay them for being such awesome friends.
Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Muhammad Arrasy Rahman)
# Table of Contents

## 1 Introduction

1.1 Ad Hoc Teamwork .................................................. 1

1.2 Open Problems in Ad Hoc Teamwork ............................. 2

1.2.1 Open Ad Hoc Teamwork ....................................... 3

1.2.2 Designing Teammate Policies for Ad Hoc Teamwork Training .................... 4

1.3 Thesis Contribution: Technical Chapters ........................ 5

1.4 List of Relevant Publications ...................................... 9

## 2 Related Work

2.1 Ad Hoc Teamwork .................................................. 13

2.1.1 Assumptions .................................................... 14

2.1.2 Related Learning Problems .................................... 16

2.1.3 Subtasks & Solution Approaches For Ad Hoc Teamwork ......... 20

2.1.4 Settings of the Ad Hoc Teamwork Problem ..................... 24

2.2 Single-Agent Reinforcement Learning ........................... 25

2.2.1 Markov Decision Process ..................................... 26

2.2.2 Value-Based Approaches ....................................... 27

2.2.3 Policy Search Approaches ..................................... 28

2.2.4 Deep Reinforcement Learning ................................ 29

2.3 Graph Neural Networks ............................................. 30

2.3.1 Input Preprocessing ............................................ 31

2.3.2 Example Computations in GNNs ................................ 32

2.4 Single-Agent Reinforcement Learning For Partially Observable Environments ........................................... 34

2.4.1 Partially Observable Markov Decision Processes ............... 35

2.4.2 Approaches For Solving POMDPs ................................ 36

2.5 Diversity in Reinforcement Learning .............................. 37
3 Problem Formulation

3.1 Open Ad Hoc Teamwork Under Full Observability
3.1.1 Open Stochastic Bayesian Games
3.1.2 Learning Objective

3.2 Partially Observable Open Ad Hoc Teamwork
3.2.1 Partially Observable Open Stochastic Bayesian Games
3.2.2 Learning Objective

3.3 Teammate Generation for Ad Hoc Teamwork Training
3.3.1 Decentralised Partially Observable Markov Decision Process
3.3.2 Learning Objective

3.4 Chapter Summary

4 Graph-based Policy Learning

4.1 General Overview

4.2 Type Inference
4.2.1 Input Preprocessing
4.2.2 Type Representation Postprocessing

4.3 Joint Action Value Modelling

4.4 Agent Modelling

4.5 Action Selection

4.6 Learning Objective

4.7 GPL Pseudocode

4.8 Complexity Analysis

4.9 Fully Observable Open Ad Hoc Teamwork Experiments
4.9.1 Experimental Setup
4.9.2 Fully Observable Open Ad Hoc Teamwork Results
4.9.3 Joint Action Value Analysis in GPL
4.9.4 Action Value Analysis in Single-Agent RL Baselines

4.10 Chapter Summary

5 Graph-based Policy Learning in Partially Observable Environments
5.3 Representation-based Latent Variable Inference .................................. 85
  5.3.1 Autoencoder Architecture ......................................................... 86
  5.3.2 Action Selection ................................................................. 87
5.4 Particle-based Latent Variable Inference .............................................. 87
  5.4.1 Belief Representation .............................................................. 87
  5.4.2 Belief Update ................................................................. 88
  5.4.3 Model Architecture .............................................................. 92
  5.4.4 Action Selection ................................................................. 94
5.5 Variational Autoencoder-based Latent Variable Inference ......................... 96
  5.5.1 Belief Representation & Update .................................................. 96
  5.5.2 Model Architecture .............................................................. 97
  5.5.3 Action Selection ................................................................. 98
5.6 Learning Objective ................................................................... 99
  5.6.1 Particle-based Belief Inference Models ............................................ 99
  5.6.2 Variational Autoencoder-based Belief Inference Models .................... 101
  5.6.3 Representation-based Models ..................................................... 102
5.7 GPL Pseudocode Under Partial Observability ........................................ 103
5.8 Partially Observable Open Ad Hoc Teamwork Experiments ....................... 104
  5.8.1 Experimental Setup .................................................................. 106
  5.8.2 Partially Observable Open Ad Hoc Teamwork Results ....................... 109
  5.8.3 Reconstruction Results .......................................................... 112
5.9 Chapter Summary ................................................................... 115

6 Generating Teammates For Ad Hoc Teamwork .......................................... 117
  6.1 Desirable Diversity For Ad Hoc Teamwork Training ......................... 118
  6.2 Best-Response Diversity Metric ..................................................... 121
  6.3 MARL-Based Diversity Optimisation ................................................. 123
    6.3.1 Data Collection .................................................................. 123
    6.3.2 Actor & Centralised Critic Architecture .................................... 124
    6.3.3 Learning Objective ............................................................. 125
  6.4 BRDiv Pseudocode .................................................................. 126
  6.5 Teammate Generation Experiments ................................................... 126
    6.5.1 Environment .................................................................. 126
    6.5.2 Teammate Generation & Evaluation Process ................................ 128
    6.5.3 Baseline Methods .................................................................. 131
### 6.5.4 Results in Collaboration With Novel Teammates

- Page 132

### 6.5.5 Generated Teammate Behaviours

- Page 133

### 6.6 Chapter Summary

- Page 135

### 7 Conclusion & Future Work

- Page 137

#### 7.1 Key Results

- Page 138

#### 7.2 Limitations of Proposed Methods

- Page 140

##### 7.2.1 GPL & Latent Variable Inference Models

- Page 140

##### 7.2.2 BRDiv

- Page 141

#### 7.3 Future Research Directions

- Page 142

### A Teammate Policies For AHT Experiments

- Page 145

#### A.1 Open AHT Experiments

- Page 145

##### A.1.1 Wolfpack

- Page 146

##### A.1.2 Level-based foraging

- Page 148

##### A.1.3 FortAttack

- Page 150

#### A.2 Teammate Generation Experiments

- Page 151

### B Hyperparameters and Network Sizes

- Page 153

#### B.1 Fully Observable Open AHT

- Page 153

#### B.2 Partially Observable Open AHT

- Page 157

#### B.3 Teammate Generation

- Page 157

### Bibliography

- Page 159
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A visualization of the forward computation process in the Relational</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Forward Model</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>The action selection process for teammates in an OSBG</td>
<td>42</td>
</tr>
<tr>
<td>3.2</td>
<td>The learner-teammate interaction in an OSBG</td>
<td>43</td>
</tr>
<tr>
<td>3.3</td>
<td>The learner-teammates interaction in a PO-OSBG</td>
<td>45</td>
</tr>
<tr>
<td>4.1</td>
<td>Overview of GPL</td>
<td>52</td>
</tr>
<tr>
<td>4.2</td>
<td>Preprocessing steps required by type inference model</td>
<td>56</td>
</tr>
<tr>
<td>4.3</td>
<td>Environment state visualisations</td>
<td>67</td>
</tr>
<tr>
<td>4.4</td>
<td>Open ad hoc teamwork results (training)</td>
<td>72</td>
</tr>
<tr>
<td>4.5</td>
<td>Shooting-related metrics for FortAttack</td>
<td>74</td>
</tr>
<tr>
<td>4.6</td>
<td>Pairwise utility analysis</td>
<td>75</td>
</tr>
<tr>
<td>4.7</td>
<td>State values for all single-agent RL baselines</td>
<td>78</td>
</tr>
<tr>
<td>5.1</td>
<td>Overview of partially observable methods</td>
<td>84</td>
</tr>
<tr>
<td>5.2</td>
<td>Preprocessing under partial observability</td>
<td>85</td>
</tr>
<tr>
<td>5.3</td>
<td>Overview of particle-based belief update</td>
<td>89</td>
</tr>
<tr>
<td>5.4</td>
<td>Overview of particle-based action selection</td>
<td>95</td>
</tr>
<tr>
<td>5.5</td>
<td>Partially observable open ad hoc teamwork results (training)</td>
<td>110</td>
</tr>
<tr>
<td>5.6</td>
<td>Action reconstruction accuracy</td>
<td>112</td>
</tr>
<tr>
<td>5.7</td>
<td>State reconstruction accuracy</td>
<td>113</td>
</tr>
<tr>
<td>5.8</td>
<td>Existence reconstruction accuracy</td>
<td>114</td>
</tr>
<tr>
<td>6.1</td>
<td>Potential teammate types generated from information-theoretic diversity maximisation</td>
<td>119</td>
</tr>
<tr>
<td>6.2</td>
<td>A more useful set of teammate types for AHT training</td>
<td>120</td>
</tr>
<tr>
<td>6.3</td>
<td>Ad hoc teamwork results</td>
<td>133</td>
</tr>
<tr>
<td>6.4</td>
<td>Cross-play matrices in Cooperative Reaching</td>
<td>134</td>
</tr>
</tbody>
</table>
6.5 Generated policy visualisations for Cooperative Reaching ........................................... 135

A.1 Generalisation performance of a QL agent trained to interact with a
   single teammate with a fixed type .................................................................................. 146

B.1 GPL and baseline architecture details for LBF and Wolfpack ................................. 154
List of Tables

4.1 Single-agent RL baselines ........................................... 70
4.2 Open AHT results (generalisation experiments) ................. 73
5.1 Evaluated belief inference algorithms ........................... 109
5.2 Partially observable open ad hoc teamwork results (testing) . 111
Chapter 1

Introduction

In this chapter, we introduce ad hoc teamwork (AHT), the main problem addressed in this thesis. Section 1.1 highlights the importance of AHT research by describing its potential applications for solving real-world decision making problems. We then summarise a subset of open challenges that must be addressed to enable applications of AHT to real-world problems in Section 1.2. Section 1.3 then provides a summary of the chapters of this thesis and their respective contribution to solving the open challenges outlined in Section 1.2. Finally, Section 1.4 completes this chapter by providing a list of our publications that report the results of this work.

1.1 Ad Hoc Teamwork

The task of designing a single agent to effectively collaborate with teammates having a diverse range of unknown policies was first introduced by Bowling and Mc-Cracken (2005). In later work, Stone et al. (2010) also called this task the *ad hoc teamwork* (AHT) problem. As a vital assumption of AHT, the designed agent must collaborate effectively with teammates without using prior coordination mechanisms such as joint training, prespecified communication protocols, or predetermined role assignment mechanisms. The assumptions underlying AHT differentiates it from other multi-agent learning problems that assume control of more than a single agent or use joint training methods.

The main challenge in AHT stems from the different agent policies that are required for effective collaboration with distinct collections of teammates. An agent designed for AHT, which in this work is also called a *learner*, must then be capable of adapting its policy according to the encountered teammates during interaction (Mirsy...
et al., 2022). In the absence of knowledge regarding teammates’ policies, a learner must adapt their policy solely based on observed teammates’ behaviour during prior interactions.

Many real-world decision making problems display the characteristics and challenges associated with AHT. In particular, tasks that require agents and humans to coexist in the same environment without knowing each other’s policies are examples where AHT approaches are of use. For instance, consider the task of designing an autonomous vehicle that must navigate through roads populated by human-controlled vehicles or autonomous vehicles designed by other companies (Albrecht et al., 2021; Hanna et al., 2021). Another example is the task of creating agents that can collaborate with humans to win a certain game (Lupu et al., 2021). In both examples, an agent designed to solve these collaborative tasks has to adjust its behaviour to optimally interact with unknown teammates without relying on joint training. Joint training is not desirable in these examples because training against humans or agents designed by other manufacturers may be expensive or even impossible.

Prior AHT approaches (Barrett and Stone, 2015; Albrecht et al., 2016a; Barrett et al., 2017; Ravula et al., 2019; Chen et al., 2020; Mirsky et al., 2022) design a learner by applying agent modelling or communication techniques to infer certain teammate-related information based on their observed behaviour. This inferred information is then used together with single-agent reinforcement learning to create an adaptive learner policy for dealing with unknown teammates. In the next section, we identify a subset of open challenges that prevent the application of existing AHT approaches to real-world decision making problems.

### 1.2 Open Problems in Ad Hoc Teamwork

Two limitations prevent the previous AHT approaches mentioned in Section 1.1 from being applicable in many real-world problems. First, previous works assume that their proposed approaches operate in *closed* AHT environments where the set of teammates remains the same during an episode of interaction with the learner. Second, prior approaches assume access to an expert-defined set of teammate policies in which interaction with the learner during training provides experiences to train the learner. These limitations provide additional open problems that must be addressed to further extend the possible applications of current AHT approaches. In the rest of this section, we focus on outlining the specific challenges arising from these open problems.
1.2. Open Problems in Ad Hoc Teamwork

1.2.1 Open Ad Hoc Teamwork

The focus of prior works on closed AHT environments prevents them from being used in scenarios where the set of teammates changes during interaction with the learner. In real-world problems, a change in the set of teammates often occurs due to agents entering or leaving the environment without prior notification. Furthermore, teammates that join the environment may adopt a wide range of policies that require different responses from the learner. For example, consider an autonomous car that must drive through a road where other vehicles may enter or leave the road at different intersections. An autonomous vehicle under this example must then adapt its policy according to the number of vehicles surrounding it alongside their respective driving styles (Albrecht et al., 2021). In this work, we refer to the changing set of teammates as open teams or environmental openness (Eck et al., 2020). We call the problem of designing an ad hoc teamwork learner in open teams the open ad hoc teamwork problem.

The open nature of a team presents additional challenges to the learner, which increases the difficulty of the ad hoc teamwork task. The first additional challenge resulting from open ad hoc teamwork is the change in the size of the observation vector resulting from teammates entering or leaving the environment. As each agent in the environment is associated with a fixed-length feature vector that provides important information for decision making, a change in the number of agents results in a change in the number of features that a learner must consider for decision making. This variable observation size prevents many function approximation models that assume fixed input sizes from being directly applicable to estimating the learner’s optimal policy. Next, the changing team composition from teammates joining or leaving the environment adds another factor that requires the learner to adapt its policy to collaborate effectively, which in closed AHT solely arise from different teammate policies. For instance, the learner may need to adopt different roles when dealing with different collections of teammate policies (Mirsky et al., 2022).

Another challenge in open AHT results from the unavailability of state information in many real-world applications of AHT. In many problems, agents only have access to observations containing partial state information. The learner must then infer the latent environment state based only on a sequence of observations. The combination of partial observability and open ad hoc teamwork provides new challenges to the learner, which now has to jointly model the effects of environment openness while also inferring the latent state of the environment. Since the actions taken by other agents
can affect the learner’s returns, the learner has to maintain a model of all existing teammates’ actions, even for unobserved teammates. Previous works have addressed the ad hoc teamwork problem under partial observability \cite{Gu et al., 2021, Ribeiro et al., 2022} but have not considered it under environment openness.

1.2.2 Designing Teammate Policies for Ad Hoc Teamwork Training

Successful applications of AHT require a robust learner that can adapt its policy to collaborate effectively with a diverse set of teammate policies encountered in real-world problems. Prior AHT approaches produce robust learners by relying on teammate policies encountered during training. Based on the learner’s interaction with teammates during training, AHT approaches learn the best response towards the different encountered teammate policies alongside unique characteristics that differentiate their behaviour. The learner then deals with unknown teammates by identifying important characteristics based on their behaviour. The identified characteristics of an unknown teammate are then utilised to estimate the optimal action for interaction by extrapolating the best response policies designed for teammates encountered during training. Prior AHT methods often use policy mixtures \cite{Albrecht et al., 2016b, Barrett et al., 2017} or neural networks \cite{Rahman et al., 2021, Papoudakis et al., 2021a, Zintgraf et al., 2021} to generalise the best responses for training teammates towards new teammates with unknown policies.

Following its role in AHT training, designing a collection of predefined training teammate types that covers the myriad of useful cooperation strategies is essential for ensuring the robustness of learners in AHT. The design of training teammates is especially crucial in AHT environments with multiple valid cooperation strategies. By only training a learner against teammate types that behave according to a subset of the valid strategies, the learner may not acquire policies for collaborating with teammates that adopt different cooperation strategies unseen during training. This issue resembles MARL agents’ failure to collaborate with other agents that were not co-trained alongside them \cite{Hu et al., 2020, Vezhnevets et al., 2020, Rahman et al., 2021}.

In previous AHT works, researchers typically design training teammate policies by relying on domain knowledge from experts or using methods to generate teammates automatically \cite{Xing et al., 2021, Lupu et al., 2021}. The reliance on experts’ domain knowledge is problematic since such information is often unavailable or difficult to elicit in many real-world problems. On the other hand, \cite{Lupu et al., 2021} highlighted
how existing teammate generation methods may produce teammate policies with superficial differences, which do not always improve the robustness of a learner when used for AHT training.

The superficial difference between teammate types generated by existing teammate generation methods results from their utilisation of information-theoretic metrics to measure diversity. The maximisation of these information-theoretic diversity metrics encourages the emergence of teammates that select different sequences of actions during interaction with the learner. However, the difference in generated teammates’ series of actions can be superficial when a similar best response policy can effectively collaborate with the different teammate policies (Lupu et al., 2021). For example, consider a robot that needs to learn how to play football. Since agents with different dribbling styles select different actions across different states, these agents will have high diversity according to information-theoretic diversity metrics and methods optimising them will likely produce such agents. Yet, training a learner against different teammates that all like to dribble with only slight differences in their dribbling style is insufficient since the learner will only learn policies to collaborate with good dribblers.

A teammate generation method that better guarantees the robustness of the learner should ideally train the learner to collaborate with teammates with different playing styles. In the previous soccer example, this corresponds to training the learner against a teammate that prefers to focus on defending, attacking or one that prefers to pass and build up play. In this case, training an AHT agent against teammate policies requiring different best responses should force the learner to then learn different policies that complement a diverse range of teammates’ collaboration strategies.

1.3 Thesis Contribution: Technical Chapters

We conclude this chapter by highlighting the different contributions of this thesis, which further pushes the boundaries of AHT research. Our thesis contributes towards existing AHT research by proposing methods to address the open challenges in AHT research outlined by Section 1.2. The different contributions of this thesis are structured into different chapters which are summarised below:
Chapter 2: Related Work

This chapter clarifies the setting and assumptions underlying AHT by positioning it relative to other related multi-agent learning problems. We then provide a summary of prior AHT research by outlining the different facets of the AHT problem alongside the progress achieved by previous works. Finally, this chapter also introduces reinforcement learning, graph neural networks and other related techniques that play a crucial role in our proposed solutions to address open challenges to AHT research provided in Section 1.2.

Chapter 3: Problem Formulation

Chapter 3 contributes toward formal learning objectives that must be optimised when addressing each open problem highlighted in Section 1.2. Within the problem formulation associated with each open problem, we first provide a formal model of the interaction between a learner and its teammates. To formalise open ad hoc teamwork, we propose the Open Stochastic Bayesian Game (OSBG) framework to model the interaction between a learner and its teammates in open ad hoc teamwork under full observability. We then also introduce Partially Observable Open Stochastic Bayesian Games (PO-OSBG) as an extension of OSBGs to open ad hoc teamwork problems under partial observability. Based on the formal models modelling the interaction between agents, we then define learning objectives that we use to measure the optimality of different solutions to each open problem addressed in this thesis.

Chapter 4: Graph-based Policy Learning

We introduce Graph-based Policy Learning (GPL), which is our proposed approach for solving open ad hoc teamwork under full state observability, in Chapter 4. GPL is based on three main components that a learner requires for effective on-the-fly collaboration with teams of variable sizes. These three components are respectively used for teammate type inference, action prediction, and joint action value modelling. The output of these three components can be combined together to estimate a learner’s optimal policy for open ad hoc teamwork, which enables it to adapt its behaviour to effectively collaborate within different team configurations. To deal with environment openness, we implement these components as Graph Neural Networks (GNNs) (Tacchetti et al., 2019; Böhmer et al., 2020) which have been demonstrated as effective function ap-
proximation models for input data with variable sizes (Jiang et al., 2019; Hamilton et al., 2017).

Chapter 4 then provides experiments that demonstrate GPL’s ability to estimate a learner’s optimal policy in open ad hoc teamwork problems under full state observability. Our results show that a GPL-based learner achieves significantly higher returns in open ad hoc teamwork than learners that use existing value-based single-agent reinforcement learning and multi-agent reinforcement learning algorithms for training. Furthermore, a GPL-based learner also achieves significantly higher returns compared to baseline methods when evaluated under an open process it has not experienced during training. Our experiments demonstrate that GPL’s significantly higher performance results from its usage of GNNs and joint action value modelling, which enables GPL to learn the effects of other teammates’ actions towards the learner. Through additional experiments in the FortAttack environment (Deka and Sycara, 2021), we empirically demonstrate that learning the effects of teammates’ actions via the joint action value model enables a learner to acquire useful behaviour from teammates.

Chapter 5: Graph-based Policy Learning in Partially Observable Environments

This chapter presents an extension of the GPL algorithm introduced in Chapter 4 to open ad hoc teamwork problems under partial observability. We specifically achieve this by combining GPL with belief inference methods that estimate the latent environment state based on the sequence of observations and actions experienced by the learner. We evaluate different belief inference methods that allow the learner to maintain representations of important latent variables for decision making, such as the environment state, teammates’ existence, and teammates’ joint actions. The belief inference methods proposed in this work are inspired by latent variable inference techniques for sequential data such as Sequential Monte-Carlo (Doucet et al., 2001b) (SMC), autoencoders (Rumelhart et al., 1985), and variational autoencoders (Kingma and Welling, 2013). We enable the proposed belief inference methods to handle variable-sized inputs resulting from openness by using GNN-based models and graph generation techniques. Additionally, implementing the belief inference model as GNNs enables these models to output graph-based representations, which allows it to be input for GPL to estimate the learner’s optimal policy under partial observability.

In this chapter, we also evaluate the performance of the proposed belief inference
models when combined with GPL to solve partially observable open ad hoc teamwork problems. Our results show that autoencoder-based belief inference models achieve significantly higher returns in the different environments used for evaluation. Further investigation into the information encoded by the belief inference models demonstrates that autoencoder-based models more accurately infer the latent state and teammates’ joint actions compared to other proposed belief inference models. An improved ability to infer the latent state and teammate joint actions enables the learner to achieve higher returns during decision making under partial observability. We also investigate the proposed belief inference models’ ability to encode the existence of unseen teammates in the environment. Our results demonstrate that the SMC-based method is significantly better at encoding unobserved teammates’ existence. However, these methods do not translate to higher returns during decision making since SMC-based representations cannot accurately represent the latent environment state and teammates’ joint actions.

Chapter 6: Cross-Play Return-based Teammate Generation

Chapter 6 presents a method that generates a diverse set of teammate policies for AHT training called Best Response Diversity (BRDiv). BRDiv improves on existing approaches that automatically generate teammate policies for training (Lupu et al., 2021; Xing et al., 2021) by preventing the emergence of teammate types with superficial differences. Instead of optimising an information-theoretic diversity metric like previous approaches, BRDiv optimises a metric based on the returns of generated teammate types when cooperating with best-response policies for other generated teammate types.

We empirically demonstrate in Chapter 6 that BRDiv prevents the emergence of teammate types having superficial differences in their behaviour by forcing the best-response policy for a teammate type to be ineffective when collaborating with other teammate types. Our experiments then compared the returns of a learner trained with teammate types generated by XPGen, a previous teammate generation approach based on trajectory diversity maximisation (Lupu et al., 2021), and ablation of BRDiv. We empirically demonstrate the robustness of a learner trained with teammate types generated by BRDiv by showing its significantly higher returns than other evaluated baselines when dealing with previously unseen teammate types.
Chapter 7: Conclusion & Future Work

We conclude this thesis by summarising the main contributions of our work in Chapter 7. Apart from the contributions of this work, this chapter also highlights the potential shortcomings of the techniques introduced in this work. We then use the identified shortcomings as a basis for our recommendations regarding potential future work to further improve the state of AHT research.

1.4 List of Relevant Publications

This thesis is a culmination of my past works in AHT research. The list of my past publications which provide the core materials of this paper is provided below:

  
  – Chapters 3 and 4.
  
  – As the main author, my role in this publication encompasses:
    * Designing a method to solve open ad hoc teamwork under full observability.
    * Designing and executing experiments to evaluate the performance of the proposed method.
    * Analysing and reporting the results of conducted experiments.

  
  – Chapter 2.
  
  – My contribution towards the completion of this publication is provided below:
    * Formulating a nomenclature to categorise existing AHT methods with other co-authors.
    * Reading several AHT publications and categorise them based on the proposed nomenclature.
* Writing a section which positions ad hoc teamwork relative to other topics in multi-agent learning.

• Rahman, A., Fosong, E., Carlucho, I., and Albrecht, S. V. (2022b). Towards robust ad hoc teamwork agents by creating diverse training teammates. In Workshop on Ad Hoc Teamwork on IJCAI ’22
  – Chapter 6.
  – My main contribution towards the completion of this publication includes:
    * Formulating a teammate generation approach for training ad hoc teamwork agents.
    * Designing experiments to evaluate the proposed teammate generation approach.
    * Reporting the results of the conducted experiments.

  – Chapters 3 and 5.
  – Currently under review in Journal of Machine Learning Research (JMLR).
  – During the process of completing this work, my role involves:
    * Formulating methods to solve open ad hoc teamwork under partial observability.
    * Designing experiments that evaluate the performance of the proposed methods.
    * Reporting and analysing the results of the experiments.

Aside from the previously mentioned publications, I also contributed to a related publication that indirectly contributes to this thesis. The details of this publication are provided below:

The proposed method in this work inspired the approach presented in Chapter 5. This publication proposed an exact belief inference method to deal with occlusion in goal recognition methods for autonomous vehicles. While the exact belief inference method in this work was made possible by certain simplifying assumptions of the problem, the methods in Chapter 5 deal with problems where the aforementioned assumptions do not hold by using approximate belief inference methods.

My contribution towards the completion of this work involves:

* Developing parts of the code base used in the method.
* Designing evaluation scenarios to test the proposed method.
* Reporting the results from the proposed evaluation scenarios.

Finally, I was a contributor in other publications that do not directly contribute to the materials in this thesis. For these publications, I assumed a more limited role during the course of the research. Details of these publications are provided below:

Chapter 2

Related Work

This chapter summarises prior Ad Hoc Teamwork (AHT) research that serves as the foundation for methods proposed in this thesis. We start by positioning the AHT problem to other related fields by discussing the assumptions unique to AHT. This discussion is followed by a description of general subtasks the existing AHT approach addresses to solve an AHT problem. We then briefly outline the different solutions adopted by AHT methods to solve the aforementioned subtasks. Finally, we provide the variations of setups displayed by existing AHT problems.

We also discuss prior works related to the different techniques used by the approaches proposed in this work. This starts with a brief summary of single-agent reinforcement learning as a technique commonly used in AHT for training a learner’s best response policy to its teammates. We then briefly outline Graph Neural Networks (GNNs), the prominent architecture used in our approach for solving open AHT. The following discussion covers prior work on belief inference, which is central to our approaches for solving AHT under partial observability. Finally, concerning our proposed approach to generating diverse teammate policies for AHT training, we briefly summarise previous works on behavioural diversity and its application in reinforcement learning.

2.1 Ad Hoc Teamwork

AHT was proposed by Bowling and McCracken (2005) and Stone et al. (2010) as the challenge of developing an agent capable of collaborating on-the-fly with teammates with unknown policies without prior coordination mechanisms. In many potential ap-

1Although the defined problem was the same, it was called impromptu teamwork in this work.
applications of AHT, agents must work in a decentralised manner with little to no time to learn effective collaboration policies with its teammates, such as when dealing with humans or agents created by different designers. With the wide range of possible behaviours displayed by teammates, an AHT agent must be prepared to adapt its policy to collaborate with any type of encountered teammates effectively. Imbuing agents with adaptive behaviour is a challenging yet essential problem that must be solved to create versatile agents that can assume many roles in many practical domains.

In this section, we provide an in-depth description of the AHT problem. Our description starts with details about the main assumptions underlying AHT. Based on its underlying assumptions, we then compare AHT with other related learning problems. This section finally ends with a summary of subtasks addressed by existing methods that solve AHT problems, alongside an overview of existing approaches for solving each subtask.

2.1.1 Assumptions

As we have alluded to in our previous descriptions of AHT, a few assumptions distinguish it from other learning problems. Further details of the assumptions underlying AHT are provided in the following sections.

2.1.1.1 No Control Over Teammates

AHT only assumes control over a single agent, which we refer to as the learner. Meanwhile, AHT does not assume control over environment properties or teammate policies, which the learner has no knowledge over. Based on environment states containing observed features of its teammates and itself, the learner is only equipped with actions that allow it to affect the state of the environment.

An optimal learner executes actions which lead toward states that facilitate effective collaboration with other teammates. For example, soccer teammates may have different pitch areas where they are most effective. An optimal AHT learner in soccer should execute actions that result in teammates obtaining the ball in areas where they are effective. Apart from changing the state of the environment, the learner can also execute actions to affect teammates’ mental state or policy so they become more effective in collaborating with the learner. This effect where the learner leads its teammates into displaying behaviours that are more conducive to collaboration is typically common when dealing with teammates that reasons (Rabinowitz et al., 2018; Wen et al., 2018).
2.1. Ad Hoc Teamwork

2.1.1.2 Collaborative Teammates

We assume that teammates encountered in an AHT environment share the same reward function as the learner. However, teammates may have additional objectives (Grosz and Kraus, 1999) that do not go against maximising the total shared reward function. These additional objectives encourage different teammates to adopt different behaviours in maximising the total shared rewards. For instance, consider a teammate striker in soccer with an additional bonus clause related to the number of goals it scores. While such a teammate still prioritises winning the match over anything else, they will prefer a playing style where they can win while scoring as many goals as possible.

For most methods that train a learner for AHT, this collaborative assumption is inconsequential to their proposed solution for ad hoc teamwork. Specifically, the learner’s decision making process proposed by these methods still functions during interaction with agents maximising different utility functions as long as similarly behaving agents are encountered during training. However, recent works (Lupu et al., 2021; Rahman et al., 2022b) that design an adaptive learner by learning to interact with a diverse generated set of teammate policies often rely on this collaborative assumption. The reliance of these methods on the collaborative assumption is to prevent the emergence of policies that cannot even solve the collaborative task of interest, which is a likely side effect of optimising diversity metrics alone.

2.1.1.3 Varying Levels of Teammate Competency

Aside from the collaborative assumption, teammates are assumed to have varying competency levels in maximising the total shared rewards (Stone et al., 2010). This competency level is assumed to be above an unknown threshold, which reflects many real-world cooperative scenarios where highly incompetent teammates, such as those acting randomly, are rarely encountered. For example, a learner may encounter teammates with varying skills in pick-up soccer, which would all aspire to win the match regardless.
2.1.4 Absence of Prior Coordination

Another important assumption in AHT is the absence of prior coordination mechanisms between the learner and its teammates. While prior coordination mechanisms encompass many different methods to ease the collaboration between agents, a commonality is that they are often not applicable to a wide range of situations. For instance, it is difficult to design prior communication protocols or role assignment mechanisms that work well in various tasks where agents must collaborate with different types of teammates. Unless a central group imposes communication or role assignment protocols that humans and agents use during collaboration, a learner relying on such protocols will have limited applicability in addressing many real-world problems.

Prior coordination mechanisms also encompass joint training of agents, highlighted as potential methods to discover conventions agents can follow during collaboration (Hu et al., 2020). However, the emergence of such collaborative conventions requires a scenario where all potentially encountered agents and humans in the environment participate in the learner’s joint training process, which cannot be done in many problems. While access to all potentially encountered agents for joint training is not always possible, note that many existing AHT methods assume access to a subset of teammates whose certain decision making attributes are known to the learner or can interact during training. AHT approaches use the knowledge gained from collaborating with this subset of teammates as a basis for decision making when encountering unknown agents.

2.1.2 Related Learning Problems

This section positions AHT against learning problems whose connection to AHT often causes confusion. For instance, advancements in multi-agent systems research have given rise to research areas that share considerable similarities with AHT. Other topics have also explored various techniques that are useful for solving AHT. Similarly, the ubiquitous nature of AHT problems also resulted in research focusing on its application for specific domains. A summary of these related topics is provided in the following section.

2.1.2.1 Partially Observable Single Agent Reinforcement Learning

The assumption that AHT controls an agent in environments where teammates’ policies are unknown makes AHT highly similar to partially observable single agent reinforcement learning (RL). Since the unknown teammate policies are essential to uphold-
2.1. Ad Hoc Teamwork

ing the Markov assumption, these teammate policies can be viewed as the unobserved part of the environment states for decision making in AHT. Although applying techniques for partially observable single agent RL to AHT problems is possible, empirical results indicate AHT methods’ inclusion of inductive bias forcing the learner to reason about teammates’ decision making process produces better adaptive learners than viewing teammates’ policies as an unobserved part of the state (Rahman et al., 2021).

2.1.2.2 Multitask Learning

Multitask learning (Caruana, 1998) is concerned with creating a single agent policy that can generalise and perform well in various tasks. Based on this goal, AHT can be viewed as a subset of multitask learning where distinct tasks correspond to effective collaboration with different sets of teammate policies. Similar to the applicability of partially observable single agent RL methods to AHT problems, multitask learning methods can also be applied to AHT problems. Nonetheless, AHT methods have an inductive bias that forces the learner to reason about teammates’ decision making process and their effects towards the learner’s objective, empirically producing better adaptive learners for AHT (Zintgraf et al., 2021).

2.1.2.3 Multi-agent Reinforcement Learning

Unlike the previously mentioned related learning problems, AHT cannot be framed as a multi-agent reinforcement learning (MARL) problem. MARL encompasses applications of reinforcement learning methods to jointly train agents whose goal is to maximise their respective rewards in each others’ presence (Busoniu et al., 2008; Hernandez-Leal et al., 2018). During the evaluation process, MARL methods assume control over the multiple agents controlled during training and evaluate their respective rewards in the presence of each other.

Based on its problem setup, MARL inherently violates multiple assumptions underlying AHT. Since MARL assumes control over teammates during training and evaluation, it violates one of AHT’s assumptions regarding no control over teammates. In addition, solutions to MARL (Lowe et al., 2017; Foerster et al., 2017; Jiang et al., 2019) are not necessarily confined to fully cooperative environments. Furthermore, MARL training inherently establishes prior coordination mechanisms that facilitate better coordination between jointly trained agents (Hu et al., 2020), which violates AHT assumption of non-existent prior coordination.
Recent works (Vezhnevets et al., 2020; Hu et al., 2020; Rahman et al., 2021) have demonstrated that many MARL methods are ineffective for AHT. These works evaluated MARL agents’ performance when dealing with teammates who may have a significantly different decision making process to teammates encountered during MARL training. It is empirically demonstrated that the excellent team performance of MARL methods with jointly trained teammates comes at the expense of poor performance when interacting with previously unseen teammates. Finally, we also show results demonstrating a select few MARL methods’ ineffectiveness for AHT in Section 4.9.2.

2.1.2.4 Ad Hoc Teaming

Ad hoc teaming concerns designing mechanisms that enable self-interested agents with different skills and preferences to collaborate and solve a common task. Existing work in this topic (Shu and Tian, 2018) approaches this problem by designing a manager agent with two main responsibilities. First, this manager can select a subset of agents in the environment to form a team. Second, the manager can design contracts which provide incentives to agents of a team to encourage them to solve a common task.

Ad hoc teaming’s problem setup highlights its main differences with AHT. While teammate incentivisation is a possible way for a learner to encourage collaboration with its teammates, a learner in AHT cannot act as a centralised entity which forces its teammates to follow a contract and form teams to solve a problem. These contracts can be seen as a coordination mechanism that agents agree to before their interaction, which violates AHT’s assumption regarding no prior coordination.

2.1.2.5 Agent Modelling

Although it does not provide a holistic solution to AHT, techniques explored in agent modelling are often considered important components to AHT methods (Barrett et al., 2017; Rahman et al., 2021; Papoudakis et al., 2021a). Agent modelling (Albrecht and Stone, 2018) is the problem of inferring characteristics of an agent’s decision making process, such as their goals, beliefs, or policies. As we alluded to in Section 1.2.2, a learner may use prior interaction with teammates encountered during training to obtain knowledge regarding inferred teammate characteristics that are useful for decision making during collaborating with unknown teammates. These useful characteristics encompass teammates’ goals, beliefs, and policies, which agent modelling is specifically helpful for inferring.
Note that agent modelling is not the same as AHT for two reasons. First, not all agent modelling methods (Rabinowitz et al., 2018) are concerned with exploring the use of their proposed agent model for effective collaboration against unknown team-mates. Second, agent modelling techniques can also be applied to interactions containing adversarial agents trying to minimize the rewards received by the learner, which violates AHT’s assumption regarding cooperative teammates.

### 2.1.2.6 Human-Agent Interaction

A potentially exciting application of AHT can be found in the human-agent/robot community. When training an agent to assist humans, it is unrealistic to jointly train an agent with all possible humans it may encounter during its operation. An agent must adapt its policy to collaborate effectively with humans not encountered during training. Prior approaches to design agents for human-agent interaction under this setting use implicit communication or forcing agents to act in a legible manner (Breazeal et al., 2005; Dragan et al., 2013).

### 2.1.2.7 Zero-shot Coordination (ZSC)

ZSC is a special case of AHT in which teammates are assumed to be maximising the same cumulative reward function that is maximised by the learner (Lupu et al., 2021; Hu et al., 2020; Bullard et al., 2020, 2021). In the evaluation setup, ZSC normally trains multiple populations of agents that maximise the same cumulative reward function. A learner is then trained by only interacting against one of the generated teammate populations. Meanwhile, this learner is evaluated based on its performance when collaborating against teammates from a population not encountered during training.

Although ZSC contributes to important techniques to solve AHT, it is a subset of AHT in which problem setup is restricted to effective collaboration with optimal teammates. Previous work (Lucas and Allen, 2022) has demonstrated that ZSC methods are suboptimal when collaborating with suboptimal teammates. For instance, such suboptimality may emerge as a result of training teammates using a reinforcement learning algorithm which is not state-of-the-art. ZSC’s limitation when dealing with suboptimal teammates may present problems when applied to AHT environments where interaction with suboptimal teammates is unavoidable.
2.1.3 Subtasks & Solution Approaches For Ad Hoc Teamwork

In our survey on ad hoc teamwork (Mirsky et al., 2022), we highlighted four subtasks that are addressed by existing methods when designing a learner for AHT. We describe these tasks in this section to aid our discussion of existing approaches for solving AHT. We specifically organise our summary of existing approaches for solving AHT in terms of their approach to solving the four tasks underlying existing AHT methods.

2.1.3.1 Knowledge Representation

In this first subtask of AHT, a designer must represent their knowledge regarding the characteristics of the environment and teammates a learner encounters during an interaction. Examples of environment characteristics that need to be considered during the knowledge representation phase include whether the environment is discrete or continuous in terms of its action and observation spaces, alongside whether the environment is static or dynamically changing due to agents’ actions. While knowledge regarding the environment is not a part of the AHT solution itself, it determines the solutions that can be used to address the remaining tasks. For instance, a designer’s knowledge of whether an AHT environment has discrete or continuous actions determines how a learner should select its actions.

Representing knowledge regarding encountered teammates is also an important subtask in AHT since a learner requires it for decision making. The solution approaches to represent teammate-related knowledge include the following methods:

**Type-based Approaches.** A teammate’s type is an abstraction of its decision making process, which is assumed to be unknown to the learner. Based on the behaviour exhibited by a teammate, type-based AHT methods store an estimate of teammates’ types. Earlier type-based AHT methods (Barrett et al., 2011; Albrecht and Ramamoorthy, 2013a; Barrett and Stone, 2015; Barrett et al., 2017) assume that only a finite number of possible teammate types exist in the environment. An expert is assumed to provide this finite set of possible teammate types. By contrast, recent type-based approaches (Rahman et al., 2021; Zintgraf et al., 2021; Papoudakis et al., 2021a) assume that a teammate’s type can be represented as a continuous fixed-length vector produced by a neural network. By representing teammates’ types as continuous representations, these type-based AHT methods relax the assumption that possible teammate types belong to a set defined by an expert.

**Experience Replay.** An alternative to representing the knowledge obtained by
interacting against teammates as types are storing the sequence of observed states. Methods that fall under this category (Chen et al., 2020) store the sequence of observed states generated from collaboration with different teammates in an experience replay.

**Task Recognition.** Methods based on task recognition store teammate-related knowledge in terms of the possible tasks that teammates can execute in the environment. As such, teammate-related knowledge are stored in form of *plays, macro actions, or options* (Sutton et al., 1999). The stored knowledge generally outlines the applicability and termination conditions of each task, alongside a sequence of low-level actions that have to be executed during each task (Wu et al., 2021).

### 2.1.3.2 Teammate Identification

Based on the learner’s knowledge regarding its teammates, the learner has to identify the characteristics of an unknown teammate during an interaction. These inferred characteristics will then serve as a basis for decision making in the next subtask. The different methods a learner may use for teammate identification are provided below:

**Type Identification.** Based on the learner’s experience interacting with teammates, early type-based AHT methods (Barrett et al., 2011; Albrecht and Ramamoorthy, 2013a; Barrett and Stone, 2015) identify teammates by computing a belief over the set of teammate types encountered during training. Meanwhile, more recent type-based AHT approaches (Rahman et al., 2021; Zintgraf et al., 2021; Papoudakis et al., 2021a) utilise recurrent neural networks (Rumelhart et al., 1985) to summarise the learner’s interaction experience with a teammate into its fixed-length type vector. Aside from the aforementioned methods that define a type as a representation of a teammate’s policy, there are methods that define types as other characteristics of the decision making process of a teammate, such as their beliefs (Rabinowitz et al., 2018; Wen et al., 2019) or goals (Carberry, 2001; Keren et al., 2021). In all type-based AHT methods, inferred teammate types are continuously updated at each step following new interaction experiences received by the learner.

**Experience Matching.** When encountering an unknown teammate, AHT methods that store teammate knowledge as an experience replay identify teammates by searching stored experiences similar to the sequence of states observed during an interaction. For example, PLASTIC-Policy (Barrett et al., 2017) achieves this by limiting the search based on the previous and currently observed state. Meanwhile, AATEAM (Chen et al., 2020) uses a sophisticated attention-based approach that measures similarity between the entire stored experiences and displayed sequence of states.
Chapter 2. Related Work

Task Identification. Methods in which knowledge representation is based on task recognition identify teammates by inferring the tasks that teammates are executing. Previous works under this category (Wu et al., 2021; Hanna et al., 2021) identify teammates’ tasks by measuring the degree to which teammates’ behaviour is suboptimal compared to the optimal behaviour required to solve a task. Specifically, a task is deemed more likely to be executed if the total cost incurred by teammates’ observed behaviour is closer to the cost required to solve the said task optimally.

2.1.3.3 Action Selection

This subtask includes a learner’s entire decision making process to decide which actions to take when collaborating against encountered teammates. In practice, the learner decides its action based on the inferred information produced during the teammate identification subtask. This section outlines alternative approaches to selecting a learner’s actions based on the inferred information from the previous subtask.

Planning. AHT methods which select a learner’s action based on planning-based approaches (Wu et al., 2011; Barrett et al., 2014; Alford et al., 2015; Malik et al., 2018; Albrecht and Stone, 2019; Eck et al., 2020; Hanna et al., 2021) utilise planners such as Monte Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006) to decide a learner’s action. MCTS is often used due to its ability to scale to environments with many agents (Eck et al., 2020). Methods under this category require teammate identification to estimate teammates’ policies for planning. During decision making, the planner decides the learner’s action by simulating the trajectories that teammates will take during an interaction, which is then used to decide the optimal action.

Expert Policy Methods. An alternative approach for decision making is to select an action based on the recommendation of an expert policy. Expert policies are used by type-based AHT methods (Chen et al., 2020; Santos et al., 2021; Barrett et al., 2017), where an expert designs a separate expert policy to deal with each teammate type encountered during training. When collaborating with teammates with unknown types, the learner executes the action prescribed by the expert policy associated with training teammate types deemed likely by the belief estimate produced during the teammate identification subtask. Compared to the planning approach, expert policy methods avoid the need to do planning for action selection, which becomes increasingly challenging in environments with continuous or large discrete action spaces. Meanwhile, the need to specify an expert policy for each potential combination of encountered teammate types is a potential impediment to its applicability to problems with many
2.1. Ad Hoc Teamwork

**Representation-based Methods.** Rather than relying on an expert policy designed for dealing with each teammate type encountered during training, representation-based methods (Rahman et al., 2021; Zintgraf et al., 2021; Papoudakis et al., 2021a) jointly learn to represent teammate types as fixed-length continuous vectors alongside a policy required to collaborate effectively with teammates characterised by any vector. Representation-based methods select actions using neural network policies, which input consists of inferred teammate type vectors and the learner’s observation. These methods relax the need to specify different expert policies for dealing with different combinations of teammates, which allows them to be applied to environments with many agents (Rahman et al., 2021).

**Leading.** When dealing with teammates that adapt to the learner’s actions, effective collaboration requires a learner to account for its actions’ effects on teammates’ policies. Several prior methods under this category assume that teammates follow a specific method to update its policy based on the learner’s actions (Agmon et al., 2014; Foerster et al., 2017). Meanwhile, Xie et al. (2021) forgo any assumption regarding knowledge over teammates’ adaptation method and instead train neural networks to produce fixed-length representations characterising teammates’ strategy and adaptation methods, based on interaction data obtained during training.

2.1.3.4 Adaptation to Changes

During its time in the environment, a learner will continuously observe information regarding the environment and its teammates. The learner will then have to adapt its behaviour according to these observed changes. This section provides adaptation methods that a learner may adopt to better collaborate with unknown teammates.

**Belief Revision.** While most type-based AHT methods assume that teammates’ types remain the same during an interaction, a learner will observe more information about its teammates the longer an interaction goes on. In most earlier works in type-based AHT (Albrecht and Ramamoorthy, 2013a; Barrett and Stone, 2015; Albrecht et al., 2016a; Barrett et al., 2017), agents update their belief over the true types of teammates by incorporating the most recent observation via an update process based on Bayes’ rule. This update process may be slow to adjust the learner’s belief in case teammates change their types during an interaction, which occurs when low probabilities are associated with a teammate’s new type. To better deal with teammates that change types, some methods attempt to detect when a teammate switches its
type (Ravula et al., 2019). Once a type switch is detected, the learner can reset its belief to a prior that does not assign low likelihood to any particular type. Other methods define priors (Santos et al., 2021) or likelihood functions (Albrecht et al., 2016a) that are less likely to produce beliefs that assign low likelihood to a particular type.

**Hypothesis Type Space Revision.** Adaptation methods under this category are designed for scenarios where the learner encounters novel teammate types not seen during training. Barrett et al. (2017) suggested a method that detects if an unknown teammate displays similar behaviour to teammate types encountered during training. If the behaviour of the unknown teammate is deemed different from previously encountered types, Barrett et al. (2017) proposed to see this teammate as having a novel type. This method then proposes a transfer learning approach to learn models that predict this novel teammate type’s behaviour.

**Representation-based Methods.** As discussed in Section 2.1.3.3, representation-based methods learn to encode teammates’ displayed behaviour into fixed-length representations. Even when the learner is not trained to interact with adaptive teammates, methods under this category (Rahman et al., 2021; Zintgraf et al., 2021; Papoudakis et al., 2021a) will simply rely on their recurrent neural network to generalise their type representation prediction to adaptive teammates. Nonetheless, it is also possible to train representation-based methods to also model teammate adaptation by letting the learner interact against adaptive teammates (Xie et al., 2021).

**Communication.** Agent communication provides another approach for adaptation. Communication generally provides agents with a way to exchange useful knowledge (Barrett et al., 2014) or even advise each other (Shvo and McIlraith, 2020). This feedback exchange facilitated through communication enables agents to better coordinate in the face of changing situations.

### 2.1.4 Settings of the Ad Hoc Teamwork Problem

This section provides different settings where the AHT problems we address in this thesis appeared. Unlike the limited settings outlined in this section, note that AHT problems exist under a broader range of settings in previous literature. For a more comprehensive coverage of other variations of AHT problem settings, we refer the reader to our AHT survey Mirsky et al. (2022).

**Environment Openness.** Under environment openness, the set of teammates a learner encounters may change due to agents entering or leaving the environment with-
out prior notification. Environment openness introduces additional challenges in action selection and adaptation due to team configuration changes. Except for works from Chandrasekaran et al. (2016), Eck et al. (2020), none of the previous approaches addresses AHT in open environments. Unlike our proposed approach in this thesis, the methods proposed by Chandrasekaran et al. (2016) and Eck et al. (2020) rely on Interactive POMDPs (I-POMDPs), whose computational complexity scales quadratically to the number of existing agents, and MCTS to model teammates’ types and decide the learner’s actions. This reliance on MCTS and the quadratic complexity of I-POMDP prevents these methods from being applied to complex problems with many agents and large action spaces.

**Partial Observability.** In many AHT problems, a learner may not have access to the full state information during decision making. In environments with partial observability, a learner must decide its action based on the sequence of observation and actions it has experienced. Recent works address the ad hoc teamwork problem under partial observability using different approaches. Gu et al. (2021) presented ODITS, a reinforcement learning-based approach that utilises an information theoretic-based regulariser to estimate proxy representations based solely on the learner’s observations. Recently, Ribeiro et al. (2022) also presented a Bayesian prediction algorithm for addressing partial observability in AHT. However, these works are limited to addressing AHT in closed teams. Unlike previous works, this thesis provides the first approach to solving open AHT problems under partial observability.

### 2.2 Single-Agent Reinforcement Learning

Single-agent reinforcement learning (RL) encompasses different approaches to learning policies that enable an agent to maximise the cumulative rewards that it receives. Rewards are scalars provided to an agent each time it chooses an action. RL enables an agent to learn a policy for achieving a goal, assuming that the agent’s reward function aligns with the goal it must achieve.

In the context of AHT research, RL is a learning technique frequently used when designing a learner. Irrespective of their action selection mechanisms outlined in Section 2.1.3.3, most methods use reinforcement learning to let a learner learn its optimal policy when collaborating with unknown teammates. AHT methods adopting planning-based action selection methods use model-based RL techniques such as MCTS (Kocsis and Szepesvári 2006) to decide the learner’s optimal actions (Wu
Chapter 2. Related Work

et al., 2011; Barrett et al., 2014; Albrecht and Stone, 2019). Some AHT methods that use expert policies for action selection (Chen et al., 2020) also design the expert policy for dealing with each teammate type via RL. Finally, meta-learning (Rahman et al., 2021; Zintgraf et al., 2021; Papoudakis et al., 2021a) and leading-based AHT approaches (Xie et al., 2021) train a learner’s policy using deep reinforcement learning based on teammates’ inferred type vectors.

Following the significant role of RL for training learners to solve AHT, we briefly discuss RL in this section. Section 2.2.1 starts our discussion by formalising the environment-agent interaction and the goal of learning in RL. We then discuss two types of model-free RL methods that we use in the methods introduced by this thesis in Section 2.2.2 and Section 2.2.3. Finally, we briefly discuss deep learning applications to extend RL to problems with large state spaces in Section 2.2.4.

2.2.1 Markov Decision Process

Most RL approaches formalise their decision making problems as a Markov Decision Process (MDP) (Bellman, 1957b). An MDP is defined as a 5-tuple, \( \langle S, A, T, R, \gamma \rangle \). In this tuple, \( S \) and \( A \) denote the set of possible environment states and agent actions. Denoting \( \Delta X \) as the set of possible probability distributions over a set \( X \), \( T : S \times A \rightarrow \Delta S \) is the transition function that determines the way an environment’s state changes according to the action of the learning agent. \( R : S \times A \times S \rightarrow \mathbb{R} \) then denotes the reward function that decides the reward given to an agent as a consequence of its actions. Note that an MDP assumes that \( T \) and \( R \) uphold the Markov assumption, which means the value of these functions only depends on the current environment state and agent action. Finally, \( \gamma \) denotes the discount rate that determines how myopic the resulting RL policies are in maximising the cumulative rewards of the agent.

The task represented by an MDP starts from an initial state \( s_0 \in S \). At any given timestep \( t \), the interaction between an agent and the environment undergoes three steps. First, the agent observes the state information \( s_t \) and decides its action \( a_t \) according to its policy, \( a_t \sim \pi(a_t|s_t) \). Second, the agent executes its selected action in the environment, \( a_t \). Third, after executing \( a_t \), an agent receives \( s_{t+1} \sim p(s_{t+1}|s_t, a_t) \) and \( R_t \) based on \( T \) and \( R \) respectively.

Solving an MDP amounts to finding an optimal policy, \( \pi^* \). Before characterising the optimal policy given components of an MDP, we first define the state-action value
function of a policy, \( \pi \), denoted below:

\[
Q_\pi(s, a) = \mathbb{E}_{a_t \sim \pi(a_t|s_t), s_{t+1} \sim T(s_t, a_t)} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right] \quad \text{s.t.} \quad s_0 = s, a_0 = a.
\]  

(2.1)

The state-action value function computes the expected discounted sum of future rewards by following \( \pi \) after executing \( a \) at the initial state \( s \). Given the above definition of the state-action value function, a policy is deemed optimal if and only if it maximises the agent’s cumulative returns in each possible state. Assuming \( \pi^* \) denotes the optimal policy, this condition is formalised as the following condition:

\[
\forall \pi, s, a, Q_{\pi^*}(s, a) \geq Q_{\pi}(s, a),
\]

(2.2)

If all components of the MDP are known, approaches based on dynamic programming, such as value iteration [Bellman 1957a] and policy iteration [Howard 1960], can be used to find the optimal policy. However, in most applications, it is impossible to have complete information about the components of the MDP. RL provides techniques to learn the optimal policy without knowing the components of an MDP by letting the agent explore the environment. In the following sections, we provide two examples of RL approaches that learn the optimal policy for solving an MDP.

### 2.2.2 Value-Based Approaches

Value-based RL approaches learn the optimal policy by computing estimates of value or state-value functions based on obtained experience. Temporal difference approaches such as Q-Learning [Watkins and Dayan 1992] and SARSA [Rummery and Niranjan 1994] iteratively update their state-action value estimates towards target values that are computed based on their current estimates. At any given timestep \( t \), the Q-Learning update under a learning rate \( \alpha \) can be computed following this update rule:

\[
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha (R_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)),
\]

(2.3)

with \( \langle s_t, a_t, r_t, s_{t+1} \rangle \) denoting the controlled agent’s experience after executing \( a_t \) at \( s_t \).

After each update to the state-action value function, value-based methods greedily update an agent’s policy to exploit actions producing the highest state-action value at any given state by always choosing them. However, this may cause agents to cease exploring actions with lower state-action values, which is problematic when the real state-action value for certain actions is underestimated. This exploration-exploitation
Chapter 2. Related Work

dilemma highlights the need for sufficient exploration to discover potentially better trajectories at the expense of occasionally choosing suboptimal actions according to the agent’s action-value function.

Value-based RL methods typically introduce randomness for the agent to explore suboptimal actions occasionally. For example, in environments with discrete action spaces, \( \varepsilon \)-greedy policies are the common choice to balance exploration and exploitation. The probability distribution of each action for an \( \varepsilon \)-greedy policy at \( s \) is provided in the following equation:

\[
\pi(a|s) = \begin{cases} 
\varepsilon/|A| & \text{if } a = \text{argmax}_a Q(s,a), \\
(1-\varepsilon)/|A| & \text{otherwise.} 
\end{cases}
\] (2.4)

During training, \( \varepsilon \) is often slowly reduced as training commences to encourage more exploitation after a sufficient number of updates.

2.2.3 Policy Search Approaches

Policy search approaches provide an alternative to value-based approaches by directly learning a policy for the controlled agent. Methods under this category typically assume that an agent’s policy is parameterised in a specific way. These methods then learn by searching for a policy parameter value, \( \theta \), which produces sample trajectories with high expected returns as indicated in the following expression:

\[
G(\theta) = \mathbb{E}_{s_0 \sim P_0(s_0), a_t \sim \pi_\theta(a_t|s_t), s_{t+1} \sim T(s_t, a_t)} \left[ \sum_{t=0}^{\infty} \gamma^t R_t \right],
\] (2.5)

with \( P_0 \) denoting the distribution over the initial state of the environment. In practice, policy search approaches can be applied in domains with discrete and continuous action spaces alike.

REINFORCE (Williams, 1992) provides one of the earliest examples of a policy search method. In REINFORCE, a parameterised policy receives \( s_t \) as input and outputs the parameters of the distribution over the agent’s action, \( p(a_t|s_t) \). REINFORCE trains the policy parameters using gradient ascent to maximise Equation 2.5. Although an exact computation of the right-hand side of Equation 2.5 is intractable in most cases, the policy gradient theorem (Williams, 1992) enables computation of an unbiased gradient estimate of the objective function, \( \nabla_\theta G(\theta) \), using sampled trajectories. According to the policy gradient theorem, the gradient of the policy parameters
2.2. Single-Agent Reinforcement Learning

can be estimated through the following expression:

\[ \nabla_{\theta} G(\theta) = \mathbb{E}_{s_0 \sim P_0(a_t \sim \pi_0(s_t), s_{t+1} \sim T(s_t, a_t))} \left[ \sum_{t=0}^{\infty} R_t \log (\pi_\theta(a_t|s_t)) \right]. \tag{2.6} \]

Policy search approaches offer an alternative when learning value functions are difficult. This is typically the case when dealing with environments with large action spaces. It may also perform better than value-based approaches when no greedy policy can optimally solve the environment, which occurs when the task cannot be formulated as an MDP. For example, this is the case when designing a policy that can only decide actions based on partial observations of the current state due to an agent’s sensor limitations. In other problems, having a greedy policy may be undesirable. For example, multi-agent learning problems in an adversarial setting often do not have greedy optimal policies since it results in a behaviour highly exploitable by other agents.

2.2.4 Deep Reinforcement Learning

Deep reinforcement learning refers to the application of neural network models as function approximators in RL. Typically, the combination between deep learning and reinforcement learning is achieved by using deep learning models as value or policy functions mentioned in Section 2.2.2 and Section 2.2.3. This combination with deep learning enables RL to scale to problems with high dimensional state spaces or action spaces. Through the usage of neural networks, deep RL methods can learn value and policy functions which generalise well to states that have not been encountered through interaction.

In value-based deep RL approaches such as Deep Q-Networks (Mnih et al., 2015) and Double Deep Q-Networks (Van Hasselt et al., 2016), the state-action values are represented as a neural network parameterised by \( \theta \). The parameters are subsequently trained to minimise the squared error between the predicted state-action values and the computed target values indicated in the following expression:

\[ L_\theta(E) = \frac{1}{2} \sum_{(s_t, a_t, r_t, s_{t+1}) \in E} (Q_\theta(s_t, a_t) - R_t - \gamma \max_{a} Q_{\theta'}(s_{t+1}, a))^2, \tag{2.7} \]

where \( E \) denotes a batch of experiences gathered by the agent and \( \theta' \) represents the parameter of a target network designed to improve the stability of the value-based deep RL learning process.

Meanwhile, deep policy search approaches such as TRPO (Schulman et al., 2015) and PPO (Schulman et al., 2017) represent their policy functions with neural networks.
Aside from the representation of an agent’s policy as a neural network, TRPO and PPO introduce modifications to Equation 2.6 to improve the stability of its learning process. Since neural network models are compatible with gradient-based optimisation, the parameters of the neural network can be trained to optimise these methods’ objective functions.

2.3 Graph Neural Networks

Graph neural networks (GNNs) (Scarselli et al., 2008; Bruna et al., 2013) are recently proposed neural network architectures which receive graphs as input. In most applications of GNNs for learning, the input graph vertices represent some entity in the learning problem, while the edges denote the relationship between the different entities. Each vertex, edge, and possibly the graph itself is associated with a fixed-length input vector, which contains features describing each entity in the graph, their relationship with each other, and all other features describing the graph respectively.

The underlying computation of GNNs transforms the graph and its underlying fixed-length input vectors into another graph, which nodes, edges, and itself are each associated with an updated fixed-length representation vector. GNNs are then trained end-to-end to minimize a specific objective function. Like any other neural network architecture, a GNN’s training process is based on gradient-based optimisation techniques.

GNNs have been used for solving a diverse domain of problems, such as molecular property prediction (Stärk et al., 2022), social influence prediction (Qiu et al., 2018), or traffic prediction (Zheng et al., 2020). Previous works highlighted that GNNs could represent and learn functions based on different entities’ relations, making it a powerful learning model for the aforementioned applications.

Aside from its ability to model the relationship between entities, another advantage gained from using GNNs is its ability to generalise well across graphs of differing structures. For instance, Huang et al. (2020) proposed a GNN-based model which produces improved average returns compared to other types of neural network architectures when controlling robots with different morphology. In other application domains, Hamilton et al. (2017) and Jiang et al. (2019) also reported similar findings demonstrating improved prediction performance resulting from using GNNs for handling inputs of variable sizes.

There are two reasons why GNNs can generalise well across varying graph struc-
2.3. Graph Neural Networks

First, the same network processes feature vectors associated with different nodes and edges in the input graph. By sharing the same model to process different edges and nodes, GNNs reduce the number of learning parameters while also avoiding the need to train new models when processing previously unseen nodes or edges. Second, GNNs use symmetric functions that can receive variable input sizes to aggregate representations during the forward computation process. This aggregation function enables the GNNs to estimate a permutation invariant function that can receive graphs with any number of nodes or edges as input.

GNNs’ ability to learn the relation between entities while processing graphs of varying sizes provides a promising approach for solving AHT. If we treat teammates as entities modelled by GNNs, learning the relation between teammates is potentially helpful to better predict teammates’ behaviour during the teammate identification sub-task discussed in Section 2.1.3. Furthermore, its ability to deal with graphs with varying numbers of nodes and edges also means that GNNs are potentially key components to improve coordination in open AHT where the number of agents in the environment may change between timesteps.

Although the many variations of GNNs make it challenging to have a formal model that can describe all GNNs, we provide details of the GNN-based Relational Forward Model (RFM) architecture proposed for agent action prediction (Tacchetti et al., 2019). The description of RFM highlights underlying computations that make GNNs valuable when solving open AHT. Finally, Chapter 4 proposes a method to solve open AHT by combining the RFM architecture alongside other GNN architectures.

2.3.1 Input Preprocessing

We start our description of the Relational Forward Model (RFM) by highlighting the preprocessing process to prepare inputs for RFM. This preprocessing step parses state information into an input graph in which each edge, node, and input graph are associated with a fixed-length feature vector. The features associated with each node in the input graph contain information about each agent represented by each node. Meanwhile, the feature of each edge informs the relationship between agents whose nodes are connected by the edge. Finally, graph features are reserved for important features for decision making that are not associated with any nodes or edges.

To further illustrate this input preprocessing step, consider the problem of using RFM to model the decision making of agents in a soccer game. In soccer, each node
and its features will correspond to an agent and their features, such as their position, orientation, or energy levels. The features of edges connecting any pair of agents can contain information about their relationship, such as whether they are from the same team or not. Finally, the input graph features encompass information that does not describe agents or the relationship between agents, such as the ball’s location or the game’s score.

2.3.2 Example Computations in GNNs

This section highlights the computations underlying the Relational Forward Model (RFM) (Tacchetti et al. 2019), a GNN architecture proposed for agent action prediction. Non-linear functions are applied to transform RFM’s input features according to the input graph. As in most GNNs (Battaglia et al. 2018), the series of computations in the RFM model can be viewed as a message-passing scheme between different nodes and vertices. Initially, the computation updates the representation of each edge. Subsequently, the updated edge representation is propagated by updating the vertex representation as a function of edges, which has the vertex as a receiver. Finally, the updated node and edge representations can be utilized to compute an updated graph representation. Generally, visualization of this process can be seen in Figure 2.1. Details of the updates to the edge, node, and graph representation are outlined in the following sections.
2.3.2.1 Edge Representation Updates

The function that updates an edge’s vector representations, $\phi_e$, in an RFM can be based on any neural network model, such as multilayer perceptrons (Rumelhart et al., 1985). The edge vector representation update is provided in the following expression:

$$ e'_k = \phi_e(\text{Concatenate}(e_k, v_{rk}, v_{sk}, u)) $$

(2.8)

where $e^k$, $v_{rk}$, $v_{sk}$, and $u$ denote the input feature of edge $k$, input features to the source and destination nodes connected by edge $k$, alongside features associated to the graph.

The same edge representation update function, $\phi_e$, is used to update the representation of all edges in the input graph. This avoids the need to define a separate update function for different nodes, which can present problems when processing input graphs with varying sizes. Specifically, this helps ensure that the edge representation update is defined adequately for different input graph structures where the number of edges in the input graph may vary.

2.3.2.2 Node Representation Updates

Before the node representation update, it is assumed that messages are sent between source and destination nodes connected by any edge in the graph. The content of these messages propagated through each edge is assumed to be the updated vector representation of the edge as defined by Equation (2.8). These propagated messages are then utilised during the node representation update process.

The first step in the node representation update process is aggregating the incoming message propagated through the different edges pointing to a vertex. Assuming that $E'_i$ are the set of edges which has node $i$ as its receiver node, the aggregated edge update for a receiver vertex $i$ can be computed by Equation (2.9)

$$ \bar{e}'_i = \rho_{e \rightarrow v}(E'_i) $$

(2.9)

Since a node can be the destination node of varying numbers of edges, only functions that can receive inputs of varying arguments are considered for $\rho_{e \rightarrow v}$. Furthermore, $\rho_{e \rightarrow v}$ must be invariant functions in terms of the different possible orderings of messages presented as input to the function. Using permutationally invariant functions for $\rho_{e \rightarrow v}$ ensures the same representation vector is produced for isomorphic input graphs (Battaglia et al., 2018). This desirable characteristic prevents the need to define a particular ordering over the nodes of an input graph. As an example of functions that
fit these criteria, element-wise sum, element-wise maximum, element-wise minimum, or element-wise average over a collection of vectors are valid aggregation functions.

After the aggregation process, the aggregated messages are combined with the current representation of the vertex and the graph to obtain the updated vertex representation. This node update process can be expressed as:

$$v'_i = \phi^v(\text{Concatenate}(\bar{e}'_i, v_i, u)).$$  \hspace{1cm} (2.10)

In the equation above, $v_i$ denotes the feature vector associated with node $i$ while $\phi^v$ represents the node update function, which can be any neural network model. In RFM’s application to action prediction, the updated node representations are used to predict the actions of the agent represented by the node through supervised learning (Tacchetti et al., 2019).

### 2.3.2.3 Graph Representation Updates

Updates to the graph vector representation require the vector representations of all nodes and edges present in the graph to be aggregated. Two additional aggregation functions are introduced for this purpose. This edge and node aggregation process is formalised in Equation 2.11.

$$\bar{e}' = \rho^{e\to u}(E'), \quad \bar{v}' = \rho^{v\to u}(V'),$$  \hspace{1cm} (2.11)

with $E'$ and $V'$ denoting the set of updated vectors for all edges and nodes in the graph. Note that candidate aggregation functions for $\rho^{e\to u}$ and $\rho^{v\to u}$ must fulfill the same criteria as functions considered for $\rho^{e\to v}$. The aggregated graph representation is finally calculated using the following expression:

$$u' = \phi^u(\text{Concatenate}(\bar{e}', \bar{v}', u)),$$  \hspace{1cm} (2.12)

where $\phi^u$ denotes a neural network model responsible for computing the updated graph vector representation.

### 2.4 Single-Agent Reinforcement Learning For Partially Observable Environments

In this section, we discuss decision making in environments where the learning agent has no access to the complete environment state, which is relevant to our work in
2.4. Single-Agent Reinforcement Learning For Partially Observable Environments

Chapter [5] about ad hoc teamwork in partially observable environments. Our discussion in this section starts by introducing the formal models of decision making problems under partial observability. We then briefly describe the concept of belief inference, the cornerstone of methods for solving decision making under partial observability.

2.4.1 Partially Observable Markov Decision Processes

Previous works in single-agent reinforcement learning have handled the challenge of decision making under partial observability. In partially observable environments, the learning agent cannot observe the state of the environment and must decide its action based on observations, which contain limited and potentially noisy information about the environment state. Under this setup, the decision making problem can no longer be formalised as an MDP.

The decision making problem in partially observable environments is instead modelled as a Partially Observable Markov Decision Process (POMDP) ([Sondik, 1971]). A POMDP is defined as a 7-tuple, \( \langle S, A, T, R, \Omega, O, \gamma \rangle \). \( S, A, T, R, \) and \( \gamma \) of a POMDP is similarly defined as their respective counterparts in an MDP. However, \( \Omega \) is introduced in a POMDP to model the possible set of observations the learning agent observes at each timestep. The observation function, \( O : S \times A \rightarrow \Delta(O) \) then dictates the observation seen by the learning agent at each timestep via a distribution, \( O(o_{t+1}|s_{t+1}, a_t) \).

The interaction between an agent and the environment under partial observability mainly proceeds like in an MDP. However, the learning agent receives an observation at each timestep instead of the environment state, which is assumed to be unknown. The learner should then solely decide their actions based on their recollection of the sequence of observations it has observed alongside its sequence of executed actions.

Solving POMDPs amounts to finding an optimal policy, \( \pi^* \), which maximises the learning agent’s returns given their history of previously observed observations and executed actions, \( H = \{o_{\leq t}, a_{\leq t}\} \). Given the components of a POMDP, the expected return of a policy \( \pi \), \( Q_\pi(H, a) \), is defined as:

\[
E_{a_T \sim \pi(a_T|H_T), \{T(s_T, a_T), o_T \sim O(o_T|s_T, a_{T-1})\}} \left[ \sum_{T=t}^{\infty} \gamma^{T-t} R(s_T, a_T) \mid H_t = H, a_t = a \right].
\] (2.13)

A policy, \( \pi^* \), is then deemed optimal if and only if:

\[
Q_{\pi^*}(H, a) \geq Q_\pi(H, a),
\] (2.14)

for all possible \( \pi, H, \) and \( a \).
Chapter 2. Related Work

2.4.2 Approaches For Solving POMDPs

Some previous works for solving POMDPs (Lin and Mitchell [1992], Whitehead and Lin [1995], Wierstra et al. [2007], Hausknecht and Stone [2015]) estimate state representations as a fixed-length vector produced by RNNs (Rumelhart et al. [1985]). These approaches provide the sequence of observations and actions experienced by an agent into an RNN that produces a state vector, which is then used as input to a policy network to decide the learner’s optimal policy. These methods use RNNs, which maintain latent states summarizing its previous inputs, to prevent agents from forgetting important decision making information derived from past observations and executed actions.

Alternative methods to solve POMDP compute probability distributions, commonly called belief states, over the set of environment states, \( B_t(s_t) = p(s_t|H_{t-1}, o_t, a_{t-1}) \). Belief state inference is motivated by the possibility that multiple states in the environment have a non-zero likelihood, given the sequence of observations and actions experienced by the learning agent (Cassandra et al., [1994]). Based on the agent’s most recent observation and action, earlier methods under this category (Cassandra et al., [1994]; Kurniawati et al., [2008]) update an agent’s belief state estimate using the Bayes theorem assuming knowledge to the state space, transition function, and observation function of the POMDP. At each timestep, this Bayesian belief update is denoted as:

\[
B_{t+1}(s_{t+1}) = \frac{O(o_{t+1}|s_{t+1}, a_t) \sum_{s_t} T(s_{t+1}|s_t, a_t) B_t(s_t)}{\sum_{s_{t+1}} O(o_{t+1}|s_{t+1}, a_t) \sum_{s_t} T(s_{t+1}|s_t, a_t) B_t(s_t)} \quad (2.15)
\]

More recent works on belief state inference (Igl et al., [2018]; Singh et al., [2021]) operate without assuming knowledge over the state space, transition function, and observation function of the solved POMDP. States are instead represented as fixed-length vectors produced by neural networks. These methods then use Sequential Monte Carlo (SMC) (Doucet et al., [2001a]), which is also called particle filters (Albrecht and Ramamoorthy, [2016]) in other literature, to update the belief defined over these state representations. Finally, these state representations are combined and used as input for optimal policy estimation under partial observability.

In this thesis, we use these different approaches to build a representation of the latent state for solving open ad hoc teamwork under partial observability. Unlike in single-agent RL, the latent variables that must be estimated in open ad hoc teamwork encompass teammates’ state features, existence, types, and previous actions. We present the application of these methods to solve open ad hoc teamwork under partial observability in Chapter 5.
2.5 Diversity in Reinforcement Learning

This section broadly discusses the potential applications of agent diversity in reinforcement learning. The first part of our discussion details the applications of diverse teammate generation in AHT. We then further discuss the potential use and approaches for diversity maximisation in other decision making problems unrelated to AHT.

2.5.1 Diverse Teammate Generation for Ad Hoc Teamwork

Previous approaches have explored diverse teammate generation methods to improve the robustness of agents when dealing with unknown teammates (Xing et al., 2021; Lupu et al., 2021; Lucas and Allen, 2022). These methods are based on the idea that interacting with a broader range of teammate behaviour during training will help a learner acquire more comprehensive cooperative strategies to deal with unknown teammates. Such teammate policy generation methods have been explored in AHT and other related problems, such as in zero-shot coordination (Hu et al., 2020).

Previous teammate generation methods for AHT formulate diversity in terms of information-theoretic measures defined over the learner’s generated trajectories (Xing et al., 2021; Lupu et al., 2021; Lucas and Allen, 2022). Denoting \( \tau \) as a potential trajectory generated by a policy, Lupu et al. (2021) attempted to generate a set of diverse teammate policies, \( \{\pi_i\}_{i=1}^{K} \), by maximizing the Jensen-Shannon divergence metric defined as:

\[
\text{JSD}(\{\pi_i\}_{i=1}^{K}) = -\frac{1}{K} \sum_{i=1}^{K} \sum_{\tau} \pi_i(\tau) \log \left( \frac{\hat{\pi}(\tau)}{\pi_i(\tau)} \right)
\]

(2.16)

, with \( \hat{\pi} \) denoting a policy that chooses its action by averaging the action likelihood of all policies in \( \{\pi_i\}_{i=1}^{K} \) at each timestep. Meanwhile, Lucas and Allen (2022) defined their diversity metric in terms of a classifier, \( q(i|\tau) \), trained to distinguish the identity of policies (\( \{1,2,\ldots,K\} \)) based on their generated trajectories. This classifier is then used to formulate a diversity metric defined as:

\[
\text{Div}(\{\pi_i\}_{i=1}^{K}) = \sum_{i=1}^{K} \mathbb{E}_{\tau \sim \pi_i} \left[ \log (q(i|\tau)) \right].
\]

(2.17)

Despite their prevalence in previous works, Lupu et al. (2021) and Liu et al. (2021) highlighted that training with teammates generated by optimising information-theoretic trajectory diversity metrics does not always lead to improved learner’s robustness. We also demonstrate this through our experiments in Chapter 6. A learner’s robustness
may not improve because policies generating distinct trajectories can be dealt with using the same best response policy during collaboration.

Liu et al. (2021) proposed the only other work that does not generate teammates based on maximising an information-theoretic diversity metric. Instead of maximising an information-theoretic diversity metric, this method proposed a diversity metric based on the best response policies’ performance. However, this work was limited to zero-sum games. In Chapter 6 we propose another method to generate training teammates for AHT based on best response policy performance.

### 2.5.2 Applications of Diversity in Other Decision Making Problems

Diversity maximisation has also been applied to single-agent and multi-agent RL. In single-agent RL, it is mainly utilised as a way for learning agents to increase exploration (Pathak et al., 2017; Hong et al., 2018; Parker-Holder et al., 2020) or discover reusable skills (Eysenbach et al., 2019). For example, Eysenbach et al. (2019) proposed a method to learn a diverse set of reusable skills by only maximising an information-theoretic objective.

The applications of diversity maximisation in multi-agent RL encompass exploration and role specialisation. MAVEN (Mahajan et al., 2019) improves exploration in multi-agent RL by maximising a mutual information metric between the trajectories and a latent space. Meanwhile, Li et al. (2021) proposed a method optimising an information-theoretic objective to facilitate agents’ specialisation towards a diverse range of roles for solving a MARL problem.
Chapter 3

Problem Formulation

We provide a detailed description of the ad hoc teamwork (AHT) problems this thesis addresses by formalising them in this chapter. The problems we address include open AHT under full observability, open AHT under partial observability, and teammate generation. The problem formulation for each addressed problem consists of two parts. First, we introduce a formal model underlying the interaction between agents in each problem. Based on the introduced formal models, we formulate a learning objective to evaluate different methods we compare in experiments associated with each problem.

This chapter is structured into three sections, each providing a problem formulation for one of the AHT problems addressed in this thesis. In Section 3.1, we provide a formulation for open AHT under full observability by introducing the Open Stochastic Bayesian Game (OSBG) model. An extension of the OSBG model to partially observable open AHT problems is then presented in Section 3.2. Finally, we formulate the teammate generation problem in Section 3.3 based on a Decentralised Partially Observable Markov Decision Process (Dec-POMDP) (Bernstein et al., 2002), which is an existing formulation for agent interaction in multi-agent systems.

3.1 Open Ad Hoc Teamwork Under Full Observability

A formal model for ad hoc teamwork (AHT) must fulfil two requirements. First, the model must formalise the interaction between agents and its effects towards the information perceived by the learner. Second, these models must represent the absence of knowledge regarding teammates’ decision making process.

The Stochastic Bayesian Game (SBG) model (Albrecht et al., 2016a) fulfils the aforementioned requirements by combining the Stochastic Game (Shapley, 1953) and
Bayesian Game model ([Harsanyi, 1967]. Using the formalism defined in stochastic games, SBG formally models the effects of the agents’ joint actions on the learner’s observed states and rewards. At the same time, SBG integrates the concept of types from Bayesian games to encapsulate the set of unknown information regarding teammates’ decision making process. While SBGs are adequate formal models for AHT problems with a fixed number of teammates, they are not suitable models for open AHT due to their inability to formalise teammates entering and leaving the environment during an interaction.

To address the limitations of the SBG model as a formal definition of open ad hoc teamwork, we introduce the Open Stochastic Bayesian Game (OSBG) model in Section 3.1.1 OSBG extends the SBG model to open environments by adding components that formalise environment openness. The learning objective that a learner must optimise when solving open AHT under full observability is then provided in Section 3.1.2.

3.1.1 Open Stochastic Bayesian Games

This section defines the Open Stochastic Bayesian Game (OSBG) model to formalise the open AHT problem. We define an OSBG as follows:

**Definition 3.1.1. OSBG.** An OSBG, \((N, S, A, \Theta, R, P, \gamma)\), is a tuple whose components are defined as follows:

- \(N\): The finite set of possible agents, \(N = \{1, 2, \ldots, n\}\).
- \(S\): The finite state space.
- \(A\): The finite set of possible actions of each agent in the environment.
- \(\Theta\): The finite set of types that teammates can assume.
- \(\gamma\): The discount rate.

Before defining the remaining components of an OSBG, we first introduce notations regarding the agent type assignment and action selection under a variable number of agents. Note that a valid action selection and type assignment restricts each agent to only be associated with a single type and action. Assuming \(\mathcal{P}(S)\), \(a_i\), and \(\theta_i\) as the power set of set \(S\), action selected by agent \(i\), and type assigned to agent \(i\) respectively, we define notations for valid agent type and action assignments as follows:
3.1. Open Ad Hoc Teamwork Under Full Observability

\[ A_N = \{ a | a \in \mathcal{P}(N \times A), \forall (i, a_i), (j, a^j) \in a : i = j \Rightarrow a_i = a^j \} \]
denotes the joint agent-action space, which is the set of all possible joint action selections under variable number of agents. The predicates that define the membership of \( a \) to \( A_N \) ensure that each agent can only select a single action in a valid joint action selection. Its elements, \( a \in A_N \), are referred as joint agent-actions.

\[ \Theta_N = \{ \theta | \theta \in \mathcal{P}(N \times \Theta), \forall (i, \theta^i), (j, \theta^j) \in \theta : i = j \Rightarrow \theta^i = \theta^j \} \]
is the joint agent-type space, which denotes the set of all possible assignments of types under variable number of agents. Similar to \( A_N \), the predicates in the membership conditions of \( \Theta_N \) ensures that each agent can only be assigned a single type. Its elements, \( \theta \in \Theta_N \), are then referred to as the joint agent-type. Note that during action selection, teammates’ types are unknown to the learner since we do not assume knowledge over teammates’ types in AHT.

With \( \Delta(X) \) denoting the set of all probability distributions over a set of random variables \( X \), the remaining components of an OSBG are defined as follows:

- \( R : S \times A_N \mapsto \mathbb{R} \), which is the reward function that determines the rewards received by the learner.

- \( P : S \times A_N \mapsto \Delta(S \times \Theta_N) \), which is the transition function which determines the next state and joint agent-types encountered by the learner, given the current state and joint agent-actions.

The interaction between a learner and its teammates in an OSBG starts from an initial state, \( s_0 \in S \). To model changing numbers of agents in the open environment, different subsets are sampled from \( N \) to model the set of existing agents in the environment at each timestep. At the beginning of the episode, the initial set of teammates, \( N_0 \subseteq N \) are sampled and assigned the joint agent-type \( \theta_0 \in \Theta_N \). The initial state \( (s_0) \), set of teammates \( (N_0) \), and joint agent-type \( (\theta_0) \), are sampled from the initial distribution \( P_0 \in \Delta(S \times \Theta_N) \).

At each timestep, the interaction between the learner and its teammates undergoes two steps. First, teammates select their respective actions according to the observed state of the environment \( s_t \) at time \( t \). Each teammate selects its action based on its current policy, \( \pi : S \times \mathcal{P}(N) \times \Theta \mapsto \Delta(A) \), conditioned on the state, the set of existing teammates, and its assigned type. This policy of a teammate, \( j \), is illustrated in Figure 3.1. Meanwhile, the learner chooses its actions based on its sequence of previously observed states and executed actions, \( H_t = \{ s_{t-1}, a_{t-1} \} \), without knowing teammates’
types or actions, which formalises the lack of knowledge regarding teammates’ decision making process assumed by ad hoc teamwork problems. Unlike its teammates, the learner chooses its actions based on $H_t$ because it has no knowledge of its teammates’ types and must infer it through their observed behaviour throughout the interaction.

![Diagram](image.png)

Figure 3.1: The action selection process for teammates in an OSBG. At each timestep, a teammate policy selects an action based on the current environment state, the set of existing teammates, and its type.

The second step occurs due to the execution of joint agent-actions chosen by agents at the first step. Following the execution of the joint agent-actions, the learner receives a scalar reward, $r_t$, which is determined by the reward function, $R: S \times A_N \mapsto \mathbb{R}$. The environment state, the set of existing teammates, and the joint agent-type all change following the transition function defined by $P: S \times \Theta_N \times A_N \mapsto \Delta(S \times \Theta_N)$. Aside from determining the next state observed by the learner, the transition function $P$ also models how teammates may enter or leave the environment by determining the teammates the learner encounters at the next timestep alongside their respective types. We assume that the learner is able to identify teammates that just left or entered the environment by comparing the set of teammates it observes currently and previously. An illustration of this process where the learner receives state and reward information after the joint agent-actions’ execution is provided in Figure 3.2.

3.1.2 Learning Objective

Solving an OSBG requires the learner, denoted by $i$, to find an optimal policy, $\pi^{i,*}$, which selects the optimal action based on the learner’s previously experienced environment states, observed teammates, and executed actions, $H_t = \{s_{\leq t}, a_{<t}\}$. We define the optimal policy as follows:

**Definition 3.1.2.** Let the joint agent-actions and the joint policy of teammates at time $t$ be denoted by $a_{t}^{-i}$ and $\pi_{t}^{-i}$, respectively. Given $0 \leq \gamma < 1$, and the learner’s previous
At any given timestep, each agent in the environment chooses their actions based on a policy conditioned on the current set of teammates and the state of the environment. Each agent then executes their actions jointly. The execution of the joint action then changes the set of existing teammates and the environment state according to the transition function. As another result of the execution of the selected joint action, the learner will also receive a reward scalar based on the reward function.

A learner’s policy, \( \pi^i \), is then optimal if and only if:

\[
\bar{Q}_{\pi^i}(H, a^i) \geq \bar{Q}_{\pi^i}(H, a^i),
\]

for all possible \( \pi^i \), \( H \), and \( a^i \). Given \( \bar{Q}_{\pi^i}(H, a^i) \), a learner’s optimal policy is to choose actions with the highest state-action value greedily. Note that to remove any ambiguity in the text, we use the bar notation \( \bar{Q}_{\pi^i} \) to denote the action-value function of the learner.

3.2 Partially Observable Open Ad Hoc Teamwork

OSBGs are inadequate to model open AHT problems under partial observability following the lack of components modelling the limited information observed by the learner. Following this limitation, Section 3.2.1 introduces the Partially Observable
Chapter 3. Problem Formulation

Open Stochastic Bayesian Game (PO-OSBG) model, which extends OSBGs to partially observable environments. Based on the PO-OSBG model, we then introduce a learning objective when solving a partially observable open AHT problem in Section 3.2.2.

3.2.1 Partially Observable Open Stochastic Bayesian Games

A Partially Observable Open Stochastic Bayesian Game (PO-OSBG) extends OSBG by introducing components which model the observations received by the learner during interaction with its teammates. We define the PO-OSBG model as follows:

Definition 3.2.1. A PO-OSBG is a 9-tuple, consisting of the following components $(N, S, A, \Theta, R, P, \Omega, O, \gamma)$. In a PO-OSBG, $N, S, A, \Theta, R,$ and $\gamma$ are defined exactly as their respective counterparts in an OSBG. The remaining components of a PO-OSBG, which model the information received by the learner under partial observability, are defined as follows:

- $\Omega$: The learner’s set of possible finite observations.
- $O : S \times N \mapsto \Delta(\Omega)$, which is the observation function that determines the distribution of observations received by the learner given the current state of the environment and the learner’s set of teammates.

In a PO-OSBG, the interaction between a learner and its teammates commences similarly to their interactions in OSBGs. The main difference is that the learner does not perceive the subsequent state information after all agents execute their respective actions. Instead, at each timestep, the learner receives an observation sampled from the distribution outputted by the observation function $O : S \times N \mapsto \Delta(\Omega)$, which is determined by the state and set of teammates that exist in the environment at $t$. This absence of state information forces the learner $i$ to choose actions based only on the sequence of observations and its actions until the present time, $H_t = \{o_{\leq t}, a_{\leq t}\}$. On the other hand, teammates receive their observations of the environment according to their observation function, which we assume as unknown information that is part of the teammate’s type. Note that viewing teammates’ unknown observation function as part of their types can be used to extend OSBGs to formalise fully observable ad hoc teamwork problems with teammates incapable of seeing the complete environment state. This interaction between agents in a PO-OSBG is illustrated in Figure 3.3.
3.2. Partially Observable Open Ad Hoc Teamwork

The learner–teammates interaction in a PO-OSBG. The colourless circles with solid borders denote the random variables observable to the learner. On the other hand, the dark-coloured circles denote the random variables that are unobserved by the learner. Finally, the dashed circles denote the joint agent-actions selected at a certain timestep, which consists of individual actions illustrated inside the connected dashed rectangle.

3.2.2 Learning Objective

Solving a PO-OSBG amounts to finding an optimal policy for action selection based on the sequence of the learner’s previous observations and executed actions, \( \pi^i_\star(o_{\leq t}, d_{<t}^i) \).

We define the optimal policy under a PO-OSBG as follows:

**Definition 3.2.2.** Let the unknown joint agent-actions and the joint policy of teammates at time \( t \) be denoted by \( a_T^{-i} \) and \( \pi_T^{-i} \), respectively. Given \( 0 \leq \gamma < 1 \), and \( H_t = \{ o_{\leq t}, d_{<t}^i \} \), the action-value of a policy \( \pi^i \) is defined as:

\[
\bar{Q}^\pi(H_t, a^i_t) = \mathbb{E}_{(s_t, h_t) \sim p(\cdot | H_t), a_T^{-i} \sim \pi_T^{-i}, (s_{T+1}, \theta_{T+1}) \sim P(\cdot | s_T, \theta_T, a_T)} \left[ \sum_{T=t}^{\infty} \gamma^{T-t} R(s_T, a_T) \mid H_t, a^i_t \right],
\]

\( (3.3) \)

A learner’s policy, \( \pi^i_\star \), is then optimal if:

\[
\bar{Q}^\pi_\star(H_t, a^i_t) \geq \bar{Q}^\pi(H_t, a^i_t),
\]

\( (3.4) \)

for all possible \( \pi^i, H_t, \) and \( a^i \). The learner’s optimal policy is to then greedily choose actions with the highest action value for any given \( H_t \) experienced by the learner.
Chapter 3. Problem Formulation

Note that we assume knowledge of state information during training in our solutions for solving open AHT under partial observability. Previous works on partially observable closed ad hoc teamwork (Papoudakis et al., 2021a; Gu et al., 2021) used this state information to train models with auxiliary loss functions to reconstruct the state. Such techniques improve the resulting returns of the learner during evaluation, where the has to interact with others without knowing the state information. Nevertheless, note that our existing PO-OSBG problem formulation and the learning objective we define are still applicable for partially observable open AHT even when state information is not known during training.

3.3 Teammate Generation for Ad Hoc Teamwork Training

In this section, we formalise the problem of teammate policy generation for training an AHT learner. We start by formalising agents’ interaction within the teammate generation problem in Section 3.3.1. Since our method and other previous work in this area (Lupu et al., 2021; Xing et al., 2021; Lucas and Allen, 2022) generate teammates using multi-agent reinforcement learning (MARL) optimisation techniques, we formulate the interaction between agents as a Decentralised Partially Observable Markov Decision Process (Dec-POMDP) (Bernstein et al., 2002), which is an existing formal model for MARL. Section 3.3.2 then formalises the main objective of a teammate generation process given a Dec-POMDP.

3.3.1 Decentralised Partially Observable Markov Decision Process

Unlike in AHT training, we do not assume access to teammates with different types during teammate generation. Instead, this teammate generation process aims to train different teammate policies, which later are followed by different teammate types in AHT training. Following this need to control and train teammate policies for teammate generation, we require a different problem formulation from OSBGs to allow teammates to be controlled by a policy we train.

We model the interaction between agents during a teammate generation process as a Decentralised Partially Observable Markov Decision Process (Dec-POMDP) (Bernstein et al., 2002). A Dec-POMDP is formally defined as follows:
Definition 3.3.1. A Dec-POMDP is a 9-tuple, $(N, S, \{A^i\}_{i=1}^{N}, P, R, O, \{\Omega^i\}_{i=1}^{N}, \gamma)$, which components are defined below:

- $N$: The finite set of possible agents, $N = \{1, 2, \ldots, n\}$.
- $S$: The finite set of possible states.
- $A^i$: The finite action space of agent $i \in N$.
- $P: S \times A^1 \times \cdots \times A^N \rightarrow \Delta S$, which denotes the transition function.
- $R: S \times A^1 \times \cdots \times A^N \rightarrow \mathbb{R}$, which is the reward function.
- $\Omega^i$: The finite set of observations that can be observed by agent $i \in N$.
- $O: S \mapsto \Delta(\Omega^1 \times \cdots \times \Omega^N)$, which denotes the observation function.
- $\gamma$: The discount rate.

An interaction between agents within a Dec-POMDP starts from an initial state $s_0 \in S$. At timestep $t$, each agent $i \in N$ receives an observation $o^i_t \sim O(s_t)$. Each agent $i \in N$ then selects an action, $a^i_t \sim \pi^i(H^i_t)$, according to a policy, $\pi^i$, conditioned on its observation-action history, $H^i_t = \{o^i_{t-1}, a^i_{t-1}\}$. After choosing their respective actions, each agent jointly executes their selected action in the environment. The execution of the joint action, $a_t = \{a^i_t\}_{i=1}^{N}$, at timestep $t$ then changes the state of the environment into $s_{t+1}$, which is sampled from the distribution outputted by the transition function, $\sim P(s_t, a_t)$. The execution of the joint action also provides each agent with a reward scalar, $R(s_t, a_t)$, based on the reward function. Note that the reward scalar is common to all agents due to the cooperative nature of AHT problems.

3.3.2 Learning Objective

The goal of the teammate generation process is to design a set of teammate policies, $\Pi^{\text{train}} = \{\pi^1, \pi^2, \ldots, \pi^K\}$, which when being used for AHT training maximise the robustness of the learner. Formalising this goal as a measurable learning objective requires a measure of robustness given a Dec-POMDP. Once a robustness measure is formally defined, a learning objective can be formulated by defining how generated teammate training policies affect a learner’s robustness.
In this thesis, we deem a learner policy as robust if it achieves high returns when collaborating with teammates from an evaluation set, \( \Pi^{eval} \). Given a Dec-POMDP and a learner policy \( \pi_i \), our proposed measure of robustness is defined below:

\[
M_{\Pi^{eval}}(\pi_i) = \mathbb{E}_{\pi^{-i} \sim U(\Pi^{eval}), a_t \sim \pi^{-i}, a_t \sim \pi^{-i}, P, O, s_t, a_t} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right],
\]  

(3.5)

where \( U(X) \) denote a uniform distribution over a set \( X \). Equation 3.5 measures the capacity of an evaluated learner policy, \( \pi_i \), to achieve high returns when dealing with teammate types from \( \Pi^{eval} \). It is important to note that \( \Pi^{eval} \) in Equation 3.5 may consist of policies not encountered during AHT training, highlighting the need for a robust learner for effective collaboration.

Since the proposed measure of robustness depends on the set of policies in \( \Pi^{eval} \), we outline two additional assumptions regarding the policies that can appear in \( \Pi^{eval} \). First, we assume that \( \Pi^{eval} \) consists of feasible teammate policies, which both optimal and suboptimal return-optimising agents have reason to use for solving the Dec-POMDP. This assumption reflects how an encounter with suboptimal teammate policies is very unlikely in many practical applications of AHT. Second, we assume that the policies in \( \Pi^{eval} \) are diverse enough such that no best response policy will be sufficient to effectively collaborate with all teammates from \( \Pi^{eval} \).

As the missing piece to formalise the goal of the teammate generation process, we now define how a set of training teammate policies affect the robustness of a learner produced by AHT methods based on \( M_{\Pi^{eval}} \). Given \( \Pi^{train} \), the objective of any AHT method is to find an optimal AHT policy, \( \pi^{*, i}(\Pi^{train}) \), defined below:

\[
\pi^{*, i}(\Pi^{train}) = \arg\max_{\pi'} \mathbb{E}_{\pi^{-i} \sim U(\Pi^{train}), a_t \sim \pi'^{-i}, a_t \sim \pi'^{-i}, P, O, s_t, a_t} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right].
\]  

(3.6)

Based on the definition of \( \pi^{*, i}(\Pi^{train}) \) and other previously defined concepts, the goal of the teammate generation process is to find an optimal set of training teammates, \( \Pi^{*, train} \), defined as:

\[
\Pi^{*, train} = \arg\max_{\Pi^{train}} M_{\Pi^{eval}}(\pi^{*, i}(\Pi^{train})).
\]  

(3.7)

While setting \( \Pi^{*, train} = \Pi^{eval} \) provides an optimal solution to the above objective, note that the teammate generation problem operates in a setup where \( \Pi^{eval} \) is unknown during training. Therefore, the main challenge in the teammate generation problem arises as a result of optimising for \( \Pi^{*, train} \) without knowing \( \Pi^{eval} \).
3.4 Chapter Summary

This chapter presents three models that formalize agent interaction during open AHT and the AHT teamwork generation process. For open AHT under full state observability, our Open Stochastic Bayesian Game (OSBG) model extends the Stochastic Bayesian Game (SBG) model Albrecht et al. (2016a) introducing formalisms to model the changing number of teammates in open environments. We then introduce the Partially Observable Open Stochastic Bayesian Games (PO-OSBG) model that extends OSBGs to partially observable open environments by adding formalisms to model the limited information received by the learner at each timestep. Finally, the agent interaction in the teammate generation problem is formalised as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) (Bernstein et al., 2002), which is a MARL formalism we use following the use of MARL-based training methods for teammate generation.

Based on the agent interaction formulation for each problem of interest, we then define an objective function to evaluate solutions to these different problems. For open ad hoc teamwork, the objective of learning is to find an optimal learner policy that maximises the learner’s returns when dealing with teammates of unknown types. Meanwhile, teammate generation aims to create a set of training teammate types that maximises the learner’s robustness during evaluation.
Chapter 4

Graph-based Policy Learning

This chapter introduces the Graph-based Policy Learning (GPL) algorithm, our proposed method for solving open ad hoc teamwork (AHT) under full observability. Section 4.1 starts by describing GPL’s main components and their general roles in estimating the learner’s optimal policy. This is followed by additional details regarding the neural network models designed to represent each GPL component in Sections 4.2, 4.3, and 4.4. Section 4.5 then discusses how GPL estimates the learner’s optimal action-value function, which is used in the learner’s action selection process, by integrating the output of the models representing each GPL component. The learning objective for training GPL’s neural network models and the overall GPL learning algorithm are outlined in Section 4.6 and 4.7 respectively. Finally, Section 4.9 provides our open ad hoc teamwork experiments, demonstrating GPL’s effectiveness in solving open ad hoc teamwork under full observability.

4.1 General Overview

AHT problems are challenging to solve because of two problems. First, a learner’s cumulative rewards depend on the actions of unknown teammates, as we alluded to in our Open Stochastic Bayesian Game (OSBG) model in Section 3.1.1. Second, another problem is experienced by the learner following its lack of knowledge regarding teammates’ types, which encompass all relevant information pertaining their decision-making process. These two problems ultimately prevent a learner from directly anticipating teammates’ actions and estimating a policy leading to optimal collaboration.

For estimating the learner’s optimal policy, GPL relies on three components for type inference, joint action value modelling, and agent modelling. These components...
Chapter 4. Graph-based Policy Learning

Figure 4.1: **Overview of GPL.** GPL’s estimation of the learner’s optimal action-value function starts from a type inference model (green box) that separately infers each teammate’s type vector for the joint-action value and the agent model. The type inference model’s components that produce type vectors for the joint-action value and agent model are parameterised by $\alpha_Q$ and $\alpha_q$ respectively. These type vectors are input to the joint-action value (red box) and agent model (blue box) parameterised by $(\beta, \delta)$ and $(\mu, \nu)$ respectively. The output from the joint action value model and the agent model are combined using Equation (4.10) to finally compute the learner’s action-value function, which is used to select the learner’s optimal action.

are implemented as neural networks and trained according to the learner’s experience interacting with its teammates. At every timestep, the way GPL combines the output of these three models to select the learner’s action is summarised in Figure 4.1.

GPL addresses the problems underlying AHT by inferring the teammates’ types as fixed-length vectors. GPL then uses the teammates’ inferred types to predict actions that these teammates will execute. Based on teammates’ predicted actions, GPL identifies the learner’s optimal action-value function to collaborate optimally with its teammates. While single-agent RL techniques provide a viable method to directly estimate the learner’s optimal policy given teammates’ identified types, we demonstrate in Section 4.9.2 that predicting teammates’ actions and reasoning about their effects on the learner returns yields higher returns in AHT.

Based on the teammates’ inferred types, the **joint action-value model** predicts the joint action-value function, which is the learner’s cumulative returns following existing agents’ joint actions. Modelling the joint action value is crucial to solving open AHT for two reasons. First, all agents’ joint actions influence the learner’s current and future rewards, following the reward and transition function definition in OSBGs. Second, estimating the joint action-value function provides better credit assignment than value-
4.1. General Overview

Based RL methods that directly model the learner’s action-value function, such as Q-Learning [Watkins and Dayan, 1992].

Joint action value estimation prevents agents from assigning too much credit to their actions when it has minimal impact on observed rewards. While it has not been utilised as often in prior AHT methods, joint action value modelling has been demonstrated to be crucial for credit assignment in previous works on MARL [Lowe et al., 2017; Foerster et al., 2018]. Section 4.9.3 shows that improved credit assignment also enables the learner to identify and learn useful behaviour from well-performing teammates.

Given an OSBG and the joint agent-action $a$, the learner’s joint-action value function given a policy $\pi$ is defined as:

$$Q_{\pi}(H, a) = \mathbb{E}_{\theta^{-i} \sim p(\cdot | H, a^{-i} \sim \pi^{-i}, a_{t}^{i} \sim \pi_{T}^{i} (\cdot | s_{t}, \theta_{T}^{i}), (s_{t+1}, \theta_{T+1}^{i}) \sim P(\cdot | s_{t}, \theta_{T}^{i}, a_{t}))} \sum_{T=t}^{\infty} \gamma^{T-t} R(s_{T}, a_{T}) | H_{t} = H, a_{T} = a$$

(4.1)

In contrast to Equation 3.1, this joint action value function denotes the learner’s expected return from executing joint agent-action $a$ after a history of previously observed states and actions $H$, assuming other agents follow $\pi^{-i}$ according to their respective types and the learner follows $\pi^{i}$. Note that this value is directly influenced by the set of existing teammates and their respective types, which determines the joint action selected by the joint teammate policy $\pi^{-i}$. GPL accounts for the influence of existing teammate types to $Q_{\pi}(H, a)$ by incorporating the inferred teammate types when estimating this value.

It is impossible to use Equation 4.1 directly to decide the learner’s optimal action. At timestep $t$, using the joint action value model to decide the learner’s optimal action requires knowledge about joint agent-actions that teammates will select, $a_{t}^{-i}$, a fact that is unknown to the learner when deciding its actions at $s_{t}$. Nonetheless, we can still use the joint action value estimate for decision-making by exploiting the following equation:

$$\tilde{Q}_{\pi}(H_{t}, a_{t}^{i}) = \mathbb{E}_{a_{t}^{-i} \sim \pi^{-i}(\cdot | s_{t}, \theta_{t}^{-i})} \left[ Q_{\pi}(H_{t}, a_{t}) \right] | a_{i}^{t} = a_{t}^{i}$$

(4.2)

which expresses the learner’s action value function in terms of $Q_{\pi}(H_{t}, a_{t})$. Equation 4.2 dictates that the learner’s action value is the expected value of $Q_{\pi}(H_{t}, a_{t})$ under the distribution of teammates’ actions. In problems with discrete possible actions, $\tilde{Q}_{\pi}(H_{t}, a_{t}^{i})$ may therefore be computed by evaluating $Q_{\pi}(H_{t}, a_{t})$ for all possible joint actions and computing their weighted average according to teammates’ joint action probability.
Equation 4.2 highlights the importance of the agent model, which is the third component of GPL. The agent model’s role is to estimate teammates’ joint action likelihood, \( \pi^{-i}(a_{t}^{-i}|s, \theta^{-i}) \). Estimating the likelihood gives the learner predictions regarding which actions will be selected by the teammates, which enables the learner to use its joint action value model to compute \( \bar{Q}_{\pi}(H_t, a_t) \). Note that the learner’s prediction regarding teammates’ actions is based on the environment state and the teammates’ inferred types. The use of state and inferred teammate types for action prediction follows from OSBG’s formulation of teammate’s decision-making process as outlined in Section 3.1.1. Definition 4.4 further details a model which predicts teammates’ actions based on the state and inferred teammates’ types.

In the following subsections, we will describe the models implemented for each component mentioned above. We start by describing the type inference model in Section 4.2, followed by a description of the joint action value model in Section 4.3 and of the agent model in Section 4.4. We then explain how the output of these models are combined for action selection in Section 4.5. Section 4.6 concludes our method description by outlining the learning objective for training the defined models based on the learner’s experience interacting with teammates.

### 4.2 Type Inference

There are three challenges in designing type inference models for open AHT. First, the learner has to accurately predict teammates’ types even when interacting with previously unseen teammates. Second, inferring teammates’ types must be done without having ground truth knowledge of their underlying types. Third, the type inference approach has to handle variable-sized inputs due to environment openness.

GPL addresses the challenges mentioned above by representing teammate types as continuous vectors computed based on a sequence of past features associated with each teammate. These vectors are produced by a recurrent neural network trained to produce similar type vectors for teammates displaying similar trajectories during interaction with the learner. As a result, the type inference model can be trained without ground truth teammate types while also generalising well against unseen teammates if the learner has previously interacted with teammates displaying similar behaviour.

The type inference model is implemented as a Long Short-Term Memory (LSTM) network \(^\text{Hochreiter and Schmidhuber, 1997}\) whose parameters are denoted by \( \alpha \). Assuming that \( \theta_{t-1} \) and \( c_{t-1} \) are the hidden and cell states of the LSTM at timestep...
The LSTM updates the type vectors following this expression:

\[ c_t, \theta_t = \text{LSTM}_\alpha(B_t, c_{t-1}, \theta_{t-1}), \]  

(4.3)

where \( B_t \) is a preprocessed input batch containing information about the system’s current state. The green box in Figure 4.1 illustrates this LSTM-based type update. After the update process, we formulate additional steps to ensure that GPL only uses type vectors of existing agents in its optimal action value estimation. Between subsequent timesteps, the type inference model removes the type vectors of teammates no longer in the environment due to environment openness. At the same time, this step adds type vectors of newly-arrived teammates.

Further details on the computational steps inside GPL’s type inference model is provided in Section 4.2.1 and 4.2.2. Section 4.2.1 covers the required preprocessing steps to produce inputs to the RNN-based type inference model. Additional details regarding the postprocessing steps to produce type representations under a changing number of agents are provided in Section 4.2.2. Figure 4.2 summarises the entire type computation process in the type inference model.

Note that the type network is trained end-to-end with the joint action-value and agent models. We specifically define two different type networks to produce type representations for joint action-value and agent modelling, which parameters are denoted by \( \alpha_Q \) and \( \alpha_q \) respectively. The type network is trained this way because an accurate teammate type representation for open AHT ideally facilitates better joint action-value and action prediction as indicated by the role of \( \theta^{-i} \) in Equations 4.1 & 4.2. We outline the role of the agent type representations for joint action-value and action prediction in Sections 4.3 & 4.4. Details of the loss functions used to train the type networks are then provided in Section 4.6.

### 4.2.1 Input Preprocessing

GPL’s input preprocessing step ensures that a type vector is computed solely based on relevant information associated with the teammate it characterises. This preprocessing step resembles RFM’s input preprocessing process defined in Section 2.3.1. We assume that there exists an expert who can separate the observed state features of individual agents into the agent input batch, \( x \). This expert then concatenates each vector in the agent feature input batch with the remaining state features not associated with any agent, \( u \), to create an input batch \( B \). Figure 4.2 illustrates this preprocessing step. This concatenation of \( x \) and \( u \) resembles RFM’s (Tacchetti et al., 2019) concatenation
of node and graph-level features before its use in the forward computation process, which we have described in Section 2.3.2.

Consider a pickup soccer environment to provide a concrete example of this first preprocessing step. An example of agent features in $x$ includes agents’ position and orientation, which are different for each agent. In contrast, an example feature in $u$ is the ball’s location, a shared value between the different agents in the environment. Using $B$ as input to the type inference model ensures that a player’s type only depends on its trajectory when moving around the pitch.

### 4.2.2 Type Representation Postprocessing

After the LSTM-based update process, GPL utilises additional processing steps to ensure that only type vectors of existing agents are used during action value estimation. Following the assumption outlined in Section 3.1.1 where the learner can identify teammates that recently left or entered the environment, GPL removes the type vectors of the teammates leaving the environment between timesteps. Type vectors of recently appearing teammates are also set to a default value of zero vectors.

We formally define this additional LSTM output processing step with Equation 4.4.

Assuming $N_t^i$ and $N_t^d$ correspond to the sets of added and removed agents at time $t$, $f_{\text{rem}}$ removes the states associated to agents leaving the environment while $f_{\text{ins}}$ inputs a zero
vector for the states associated to agents joining the environment. This preprocessing step is illustrated by the computational steps between the two LSTM blocks visualised in Figure 4.2b.

\[
\text{Prep}(\theta_t, c_t) = f_{\text{ins}}(f_{\text{rem}}(\theta_t, c_t, N_t^{d}), N_t^{j})
\]

(4.4)

4.3 Joint Action Value Modelling

A joint action value model for open AHT must fulfil three main requirements. First, the model must be capable of handling inputs of variable sizes resulting from environment openness. Second, it must facilitate an efficient computation of the learner’s action value function based on Equation 4.2. Third, the model must also estimate the effects of teammates’ actions towards the learner’s returns.

One way to fulfil the previously mentioned requirements is to represent the joint action value model as a fully connected Coordination Graph (CG) (Guestrin et al., 2002). CGs facilitate the factorisation of joint action value functions into singular and pairwise utility terms, which we demonstrate in Section 4.5 to enable a more efficient action value computation. Implementation of CG models can also be based on GNNs (Boehmer et al., 2020), which are designed to handle inputs of variable sizes. Finally, CG’s joint action value factorisation also enables modelling the effects of teammates’ individual and pairwise actions on the learner’s returns, as demonstrated in Section 4.9.

Given a history of past states and actions from the learner, \(H_t\), and a set of existing agents at timestep \(t\), \(N_t\), a fully connected CG factorises the learner’s joint action value into the sum of singular utility terms, \(Q^{j}_{\pi}(a^j_t|H_t)\), and pairwise utility terms, \(Q^{j,k}_{\pi}(a^j_t, a^k_t|H_t)\). The joint action value factorisation for a fully connected CG follows this equation:

\[
Q^{\pi}_{\pi}(H_t, a_t) = \sum_{j \in N_t} Q^{j}_{\pi}(a^j_t|H_t) + \sum_{j,k \in N_t, j \neq k} Q^{j,k}_{\pi}(a^j_t, a^k_t|H_t).
\]

(4.5)

In terms of the contributions towards the learner’s returns, \(Q^{j}_{\pi}(a^j_t|H_t)\) can be viewed as the contribution of agent \(j\)’s action \(a^j\), while \(Q^{j,k}_{\pi}(a^j_t, a^k_t|H_t)\) is the contribution of agents \(j\) and \(k\) jointly choosing \(a^j\) and \(a^k\) respectively.

To enable generalisation across different input \(H_t\), \(Q^{j}_{\pi}(a^j_t|H_t)\) and \(Q^{j,k}_{\pi}(a^j_t, a^k_t|H_t)\) are implemented as multilayer perceptrons (MLPs) parameterised by \(\beta\) and \(\delta\) respectively. For two reasons, both models that compute the singular and pairwise utilities...
receive input solely consisting of agents’ type representations outputted by the type inference network instead of \( H_t \). First, the type inference network’s output contains information regarding the teammates’ unknown teammate types, which inference is why we had to utilize \( H_t \) for decision making in the first place. Second, it also contains important information on \( s_t \) since \( s_t \) is used as input for the type inference model. We define the types as \( \theta^i_t \) and \( \theta^j_t \), where \( \theta^j_t \) is the type vector associated to the learner and \( \theta^i_t \) is the type vector of agent \( j \).

We now define how agents’ type representations are used for the joint action-value estimation. Given the types vectors as input, MLP\(_\beta\) outputs a vector with a length of \(|A|\) that estimates \( Q_{\beta,\alpha}^j(a^j|s_t) \) for each possible action of \( j \) following:

\[
Q_{\beta,\alpha}^j(a^j|H_t) = MLP_{\beta}(\theta^j_t, \theta^i_t)(a^j). \tag{4.6}
\]

Instead of outputting the pairwise utility for the \(|A| \times |A|\) possible pairwise actions of agent \( j \) and \( k \), MLP\(_\delta\) outputs an \( K \times |A| \) matrix (\( K \ll |A| \)) given its type vector inputs. MLP\(_\delta\) computes its output matrix solely based on the type vectors, following the same reasoning as MLP\(_\beta\). Assuming a low-rank factorisation of the pairwise utility terms, the output of MLP\(_\delta\) is used to compute \( Q_{\delta,\alpha}^{j,k}(a^j_t,a^k_t|H_t) \) with the following equation:

\[
Q_{\delta,\alpha}^{j,k}(a^j_t,a^k_t|s_t) = (MLP_{\delta}(\theta^j_t, \theta^i_t))^\top MLP_{\delta}(\theta^k_t, \theta^i_t))(a^j_t,a^k_t). \tag{4.7}
\]

Previous work from Zhou et al. (2019) demonstrated that low-rank factorization enables scalable pairwise utility computation even under thousands of possible pairwise actions. Finally, note that we use the same parameters for MLP\(_\beta\) and MLP\(_\delta\) for modelling each utility term associated with teammates or a pair of teammates to encourage knowledge reuse for utility term computation. We show the importance of knowledge reuse via parameter sharing to GPL’s performance in Section 4.9.3.

Details of the reinforcement learning-based process to train GPL’s CG-based joint action value model are provided in Sections 4.6 and 4.7. Section 4.6 specifically discusses a value-based reinforcement learning objective optimized to train the joint action value model. Meanwhile, Section 4.7 outlines the learner’s experience-gathering process to obtain data for training this model.

### 4.4 Agent Modelling

Due to environment openness, the agent model has to efficiently predict the joint agent-action probability, \( \pi^{-i}(.|s_t, \theta^i_t) \), of a variable number of unknown teammates. GPL
addresses this issue by implementing the agent model as a Relational Forward Model (RFM), which we introduced in Section 2.3. We show in Section 4.5 that this GNN-based agent model facilitates an efficient computation of the learner’s estimated optimal action-value function.

While GPL’s agent model assumes that teammates choose their actions independently, it models teammates’ potential effect on each other using the Relational Forward Model (RFM) architecture (Tacchetti et al., 2019). As in other GNN models, RFM contains message-passing operations, enabling improved reasoning regarding the relationship between nodes in a graph. Furthermore, RFM has been demonstrated to provide an accurate prediction of agents’ actions likelihood (Tacchetti et al., 2019).

Following the same reasoning as in MLP\(_\beta\) and MLP\(_\delta\), the RFM-based agent model only receives agents’ type representations produced by LSTM\(_\alpha\) as input. Assuming a fully connected input graph structure, the inferred type representations of existing agents, \(\theta_t\), are treated as node features to compute a fixed-length embedding for each agent \(j \in N_t, \bar{\eta}_j\). Note that we assume a fully connected graph structure to avoid specifying domain knowledge regarding the relationship between agents.

Let \(a^j\) be the action taken by agent \(j\), we use each agent’s updated embedding to approximate \(\pi^{-i}(a^{-i}|s, \theta_{-i}^{-i})\) as:

\[
\pi^{-i}(a^{-i}|s, \theta_{-i}^{-i}) = \prod_{j \in N_t - \{i\}} q_{\zeta, \eta, \alpha}(a^j|s),
\]

with,

\[
\bar{\eta}_j = (\text{GNN}_{\zeta}(\theta_t))_j, \quad q_{\zeta, \eta, \alpha}(a^j|s) = \text{Softmax}(\text{MLP}_{\eta}(\bar{\eta}_j))(a^j).
\]

In the above equation, the GNN and MLP parameters that transform the updated agent embeddings are denoted by \(\zeta\) and \(\eta\). The forward computation process inside the GNN model proceeds according to the process we detailed in Section 2.3.

### 4.5 Action Selection

Computing the exact value of Equation 4.2 for action selection can be challenging in many practical applications. For instance, a team of \(k\) agents which may choose from \(n\) possible actions requires the evaluation of \(n^k\) joint-action terms. This exponential increase in the number of evaluated terms makes the evaluation of Equation 4.2 unfeasible for large teams.
GPL reduces the computational complexity of evaluating Equation 4.2 by factorising $Q_{\pi_i}(H_t, a_t)$ and $\pi^{-i}(a^{-i}|s, \theta_{-i})$ into terms defined in Equation 4.5 and 4.8. Factorising these measures into singular and pairwise terms results in an action-value function that is also factorised into simpler terms. Substituting the joint-action value and agent models from Equation 4.5 and 4.8 into Equation 4.2 results in an action-value function with the following expression:

$$\tilde{Q}(H_t, a'_t) = Q^i_{\pi_i, \alpha_i}(a'_t|H_t)$$

$$+ \sum_{j \in N_i \setminus \{i\}, a'_t \in A_j} (Q^j_{\pi_j, \alpha_j}(a'_t|H_t) + Q^{j,j}_{\delta_j, \alpha_j}(a'_t, a'_j|H_t))q_{j, \eta_j, \alpha_j}(a'_j|s_t)$$

$$+ \sum_{j,k \in N_i \setminus \{i\}, j \neq k, a'_t \in A_j, a'_k \in A_k} Q^{j,k}_{\delta_j, \alpha_j}(a'_t, a'_k|H_t)q_{j, \eta_j, \alpha_j}(a'_j|s_t)q_{k, \eta_k, \alpha_k}(a'_k|s_t).$$

(4.10)

**Proof of Equation 4.10** By substituting Equation 4.5 and 4.8 into Equation 4.2, we can derive the following expression:

$$\tilde{Q}_{\pi_i}(H_t, a'_t) = \mathbb{E}_{a_i^t \sim \pi^{-i}(\cdot|s_t, \theta_{-i})} \left[ Q_{\pi_i}(H_t, a) \Bigg| a' = a'_t \right]$$

$$= \sum_{a' \in A_i} Q_{\pi_i}(H_t, a)\pi^{-i}(a^{-i}|s_t, \theta_{-i})$$

$$= \sum_{a' \in A_i} (Q^i_{\pi_i, \alpha_i}(a'_t|H_t) + \sum_{a'_t \in A_j} Q^{j,j}_{\delta_j, \alpha_j}(a'_t, a'_j|H_t))q_{j, \eta_j, \alpha_j}(a^{-j}|s_t, a'_t)$$

$$= Q^i_{\pi_i, \alpha_i}(a'_t|H_t) + \sum_{j \in N_i \setminus \{i\}, a'_t \in A_j} (Q^j_{\pi_j, \alpha_j}(a'_t|H_t) + Q^{j,j}_{\delta_j, \alpha_j}(a'_t, a'_j|H_t))q_{j, \eta_j, \alpha_j}(a'_j|s_t)$$

$$+ \sum_{j,k \in N_i \setminus \{i\}, j \neq k, a'_t \in A_j, a'_k \in A_k} Q^{j,k}_{\delta_j, \alpha_j}(a'_t, a'_k|H_t)q_{j, \eta_j, \alpha_j}(a'_j|s_t)q_{k, \eta_k, \alpha_k}(a'_k|s_t).$$

(4.11)

Unlike Equation 4.2, Equation 4.10 is defined in terms of singular and pairwise action terms. This limits the number of computed terms to only increase quadratically as the team size increases. Furthermore, the computation of the singular and pairwise terms in Equation 4.10 can be efficiently done in parallel with existing GNN libraries (Wang et al., 2019).

Note that a few terms from Equation 4.10 can be ignored during the action selection process. The third line in Equation 4.10 is a constant that has the same value for any
4.6 Learning Objective

Optimising GPL’s models requires interaction experiences that the learner collects. We assume the learner collects these experiences using an $\epsilon$-greedy action selection policy with its action value computation method described in Section 4.5. Given a batch of interaction experiences $D = \{(H^n_t, a^n_t, r^n_t, H^n_{t+1})\}_{1}^{D}$ with $H_t = \{s < t, a < t\}$, the agent modelling network and the type inference network providing it input is trained to estimate $\pi(a^i_t | s_t, a^i_t)$ through supervised learning by minimising the negative log-likelihood loss, which is defined below:

\[
L_{q, \zeta, \eta}(D) = \sum_{(H_t, a_t, r_t, H_{t+1}) \in D} \left( \sum_{j \in -i} \log (q_{\zeta, \eta, a_j}(a^i_t | s_t)) \right).
\] (4.12)

Also, the collected data set is used to update GPL’s joint-action value network using value-based reinforcement learning. Unlike standard value-based deep reinforcement learning approaches (Mnih et al., 2015), we use the joint action value as the predicted value. The loss function for the joint action value network is defined as:

\[
L_{\beta, \delta, \alpha_Q}(D) = \sum_{(H_t, a_t, r_t, H_{t+1}) \in D} \left( \frac{1}{2} \left( Q_{\beta, \delta, \alpha_Q}(H_t, a_t) - y(r_t, H_{t+1}) \right)^2 \right),
\] (4.13)

with $y(r_t, H_{t+1})$ being a target value which depends on the algorithm being used. We train GPL with Q-Learning (GPL-Q) (Watkins and Dayan, 1992) and Soft-Policy Iteration (GPL-SPI) (Haarnoja et al., 2018), which produces a greedy and stochastic policy, respectively. Denoting $\tilde{Q}'$ as action values computed by a target value network, the target value computations of GPL-Q and GPL-SPI are defined as the following:

\[
y_{QL}(r_t, H_{t+1}) = r_t + \gamma \max_d, \tilde{Q}'(H_{t+1}, d^i),
\] (4.14)

\[
y_{SPI}(r_t, H_{t+1}) = r_t + \gamma \sum_{d'} p_{SPI}(a^i_t | H_{t+1}) \tilde{Q}'(H_{t+1}, d') \nonumber.
\] (4.15)

GPL-SPI’s target values in Equation (4.15) assume that the learner’s policy selects actions using the following expression:

\[
p_{SPI}(a^i_t | H_t) \propto \exp \left( \frac{\tilde{Q}(H_t, a^i_t)}{\tau} \right)
\] (4.16)

with $\tau$ being the temperature parameter.
Algorithm 1 GPL Action Value Computation

Input:

Current state, \(s_t\).

Joint-action value model parameters, \((\alpha_Q, \beta, \delta)\).

Agent model parameters, \((\alpha_q, \eta, \zeta)\).

Agent model LSTM hidden vectors, \(h_{t-1,q}\).

Joint-action value model LSTM hidden vectors, \(h_{t-1,Q}\).

1: \textbf{function } QV(s_t, \alpha_Q, \alpha_q, \beta, \eta, \zeta, h_{t-1,Q}, h_{t-1,q})
2: \hspace{1em} B, \theta_Q, c_Q \leftarrow \text{PREPROCESS}(s_t, h_{t-1,Q})
3: \hspace{1em} B, \theta_q, c_q \leftarrow \text{PREPROCESS}(s_t, h_{t-1,q})
4: \hspace{1em} \theta_Q', c_Q' \leftarrow \text{LSTM}_{\alpha_Q}(B, \theta_Q, c_Q)
5: \hspace{1em} \theta_q', c_q' \leftarrow \text{LSTM}_{\alpha_q}(B, \theta_q, c_q)
6: \forall j, \bar{n}_j \leftarrow (\text{GNN}_{\zeta}(\theta_q', c_q'))_j
7: \forall j, q_{\eta,\zeta,a_q(.|s_t)} \leftarrow \text{Softmax}(\text{MLP}_{\eta}(\bar{n}_j))
8: \forall j, a^l, Q_{\beta,a_Q}(a^l|H_t) \leftarrow \text{MLP}_{\beta}(\theta_{Q}', \theta_Q')(a^l)
9: \forall j, a^l, a^k,
10: \hspace{1em} Q_{\delta,a_Q}^{l,k}(a^l,a^k|H_t) \leftarrow \text{MLP}_{\delta}(\theta_{Q}', \theta_Q', \theta_Q')(a^l,a^k)
11: \text{Compute } \bar{Q}(H_t, a^l) \text{ using Equation (4.10)}
12: \bar{Q}(H_t, .) \leftarrow \text{MARGINALIZE}(q_{\eta,\zeta,a_q(.|s_t)}, Q_{\beta}(.,|H_t), Q_{\delta}(.,.|H_t))
13: \textbf{return } \bar{Q}(H_t, .), (\theta_Q', c_Q'), (\theta_q', c_q')
14: \textbf{end function}

4.7 GPL Pseudocode

As the first step to describe the complete training process underlying GPL, we first introduce the pseudocode behind the \textbf{QV} function, highlighting the entire process of estimating the learner’s action-value function starting from input preprocessing until the combination of the joint-action value and agent model outputs. The observation and hidden vector preprocessing method described in Section 4.2.1 is represented by the \textbf{PREPROCESS} function in this section. Furthermore, we denote the joint-action value and action-value computation through Equation 4.5 and Equation 4.10 as the \textbf{JOINTACTEVAL} and \textbf{MARGINALIZE} functions respectively. Algorithm 1 then provides the \textbf{QV} function, which combines the previously mentioned function to compute the learner’s estimated optimal action-value function.
Aside from these functions, we define \( \text{QJOINT} \) and \( \text{PTEAM} \), which output is required to compute the loss functions, \( L_{\beta,\delta,a_Q} \) and \( L_{\eta,\zeta,a_q} \), in Equations 4.13 and 4.12. \( \text{QJOINT} \) is a function that computes the predicted joint action value for an observed state and joint actions. Meanwhile, \( \text{PTEAM} \) computes the joint teammate action probability at a state. Both \( \text{QJOINT} \) and \( \text{PTEAM} \) are further defined in Algorithm 2 and 3.

**Algorithm 2** GPL Joint-Action Value Computation

**Input:**
- State, \( s_t \).
- Observed joint action, \( a_t \).
- Joint-action value model parameters, \((\alpha_Q, \beta, \delta)\).
- Joint-action value model LSTM hidden vectors, \( h_{t-1,Q} \).

1: \( \text{function QJOINT}(s_t, a_t, \alpha_Q, \beta, \delta, h_{t-1,Q}) \)
2: \( B, \theta_Q, c_Q \leftarrow \text{PREPROCESS}(s_t, h_{t-1,Q}) \)
3: \( \theta_Q', c_Q' \leftarrow \text{LSTM}_{\alpha_Q}(B, \theta_Q, c_Q) \)
4: \( \forall j, a^j, Q^j_{\beta,a_Q}(a^j|H_t) \leftarrow \text{MLP}_{\beta}(\theta_Q', \theta_Q')(a^j) \)
5: \( \forall j, a^j, a^k, Q^j_k(a^j, a^k|H_t) \leftarrow \text{MLP}_{\delta}(\theta_Q', \theta_Q', \theta_Q')(a^j, a^k) \)
6: \( Q(H_t, a_t) \leftarrow \text{JOINTACTEVAL}(Q_{\beta,a_Q}(\cdot|s_t), Q_{\delta,a_Q}(\cdot, \cdot|s_t)) \quad \triangleright \text{Compute} 
Q(H_t, a_t) \text{ using Equation 4.5} \)
7: \( \text{return } Q(s, a) \)
8: \( \text{end function} \)

We finally describe GPL’s training algorithm using the previously defined functions. GPL collects experience from parallel environments through the modified Asynchronous Q-Learning framework ([Mnih et al., 2016](#)) where we replace the asynchronous data collection process with synchronous data collection instead. Despite this, it is relatively straightforward to modify the pseudocode to use an experience replay instead of a synchronous process for data collection. As in the case of existing deep value-based RL approaches, we also use a separate target network whose parameters are periodically copied from the joint action value model to compute the target values required for optimising Equation 4.13. We finally update the model parameters in the pseudocode to optimise the loss function provided in Section 4.6 using gradient descent. GPL’s training process is finally described in Algorithm 4.
Algorithm 3 GPL Teammate Action Probability Computation

Input:
State, \( s_t \).
Observed joint actions, \( a_t \).
Agent model parameters, \( (\alpha_q, \eta, \zeta) \).
Agent model LSTM hidden vectors, \( h_{t-1,q} \).

1: function PTEAM(\( s_t, a_t, \alpha_q, \eta, \zeta, h_{t-1,q} \))
2: \( B, \theta_q, c_q \leftarrow \text{PREPROCESS}(s_t, h_{t-1,q}) \)
3: \( \theta'_q, c'_q \leftarrow \text{LSTM}_{\alpha_q}(B, \theta_q, c_q) \)
4: \( \forall j, \bar{n}_j \leftarrow (\text{RFM}_{\zeta}(\theta'_q, c'_q))_j \)
5: \( \forall j, q^{\bar{n}_j}_{\eta, \zeta, \alpha_q}(.,|s_t) \leftarrow \text{Softmax}(\text{MLP}_{\eta}(\bar{n}_j)) \)
6: \( q_{\eta, \zeta, \alpha_q}(a^{-i}|s_t, a^i_t) \leftarrow \prod_{j \in -i} q^{\bar{n}_j}_{\eta, \zeta}(a^i_j|s_t) \)
7: return \( q_{\eta, \zeta, \alpha_q}(a^{-i}|s_t, a^i_t) \)
8: end function

4.8 Complexity Analysis

The algorithmic complexity of the GPL algorithm introduced in Section 4.7 is dominated by evaluating the third term of Equation 4.10 at each timestep. Assuming \(|B|\) denotes the size of state data used for training at each timestep, this third term is evaluated for at most \(|B|(|N|^2 - |N|)\) times since it must be evaluated for every possible edge in a batch. Assuming we train GPL for \( T \) timesteps, the overall complexity of the GPL algorithm becomes \( O(T|B||N|^2A^2) \) since evaluating each the third term from Equation 4.10 requires the summation over \(|A|^2\) terms.

This overall complexity can be ameliorated by using GNN libraries [Wang et al., 2019] when implementing GPL. These libraries parallelise the computation of operations defined over nodes and edges of a batch of input graphs. Thus, the actual complexity of the GPL algorithm is just \( O(T|A|^2) \) and does not rely on the state space size of the problem.

Despite this relatively low algorithmic complexity, GPL will require lots of training experience following its reliance on model-free RL methods. Model-free RL methods require more training experience from not having models of the environment transition function. Instead of having an accurate gradient estimate for policy or value function updates by relying on knowledge of the environment’s transition function, model-free RL methods must make do with high-variance gradient estimates computed based on
Algorithm 4 GPL Training

1: Initialise the joint-action value model parameters, $\alpha_Q, \beta, \delta$.
2: Initialise the agent model parameters, $\alpha_q, \eta, \zeta$.
3: Create target joint-action value networks.
4: $\alpha'_Q, \beta', \delta' \leftarrow \alpha_Q, \beta, \delta$
5: $\theta_Q, c_Q, \theta_Q^{\text{arg}}, c_Q^{\text{arg}}, \theta_q, c_q \leftarrow 0, 0, 0, 0, 0$
6: Reset the gradients to each of GPL’s model parameters.
7: $d\alpha_Q, d\alpha_q, d\beta, d\delta, d\eta, d\zeta \leftarrow 0, 0, 0, 0, 0$
8: Observe $s_0$ from environment
9: for $t = 0$ to $T$ do
10: \begin{align*}
h_Q, h_q, h_Q^{\text{arg}} &\leftarrow (\theta_Q, c_Q), (\theta_q, c_q), (\theta_Q^{\text{arg}}, c_Q^{\text{arg}}) \\
\Qhat(H_t, \cdot), h_Q', h_Q^{\text{arg}} &\leftarrow \text{QV}(s_t, \alpha_Q, \alpha_q, \beta, \delta, \eta, \zeta, h_Q, h_q)
\end{align*}
11: Sample action according to the learning algorithm being used,
12: $a_t^i \sim \begin{cases} 
\text{eps-greedy}(e, \Qhat(H_t, \cdot)), & \text{if Q-Learning} \\
\text{SPI}(\Qhat(H_t, \cdot), \tau) & \text{if SPI}
\end{cases}$
13: Execute $a_t^i$ and observe $a_t^{-i}, r_t$ and $s_{t+1}$.
14: $Q_{\beta, \delta, \alpha_q}(H_t, a_t) \leftarrow \text{QJOINT}(s_t, a_t, \alpha_Q, \beta, \delta, h_Q)$
15: $\Qhat'(H_{t+1}, a_{t+1}^i), h_Q'^{\text{arg}} \leftarrow \text{QV}(s_{t+1}, \alpha_Q', \beta', \delta', \eta, \zeta, h_Q^{\text{arg}}, h_Q')$
16: Compute target value for updating the joint-action value model with,
17: $y(r_t, H_{t+1}) = \begin{cases} 
r + \gamma \max_{a'} \Qhat'(H_{t+1}, a'), & \text{if Q-Learning,} \\
\gamma \text{SPI}(a_t|H_{t+1})\Qhat'(H_{t+1}, a_t), & \text{if SPI}
\end{cases}$
18: Compute predicted action probabilities of teammates using the agent models,
19: $q_{\eta, \zeta, \alpha_q}(a_t^{-i}|s_t, a_t^i) \leftarrow \text{PTEAM}(s_t, a, \alpha_q, \eta, \zeta, h_q)$
20: Compute $L_{\zeta, \eta, \alpha_q}$ and $L_{\beta, \delta, \alpha_Q}$ based on Equation 4.12 and 4.13.
21: $d\alpha_Q = d\alpha_Q + \nabla_{\alpha_Q} L_{\beta, \delta}, d\alpha_q = d\alpha_q + \nabla_{\alpha_q} L_{\eta, \zeta}$
22: $d\beta = d\beta + \nabla_{\beta} L_{\beta, \delta}, d\delta = d\delta + \nabla_{\delta} L_{\beta, \delta}$
23: $d\eta = d\eta + \nabla_{\eta} L_{\eta, \zeta}, d\zeta = d\zeta + \nabla_{\zeta} L_{\eta, \zeta}$
4.9 Fully Observable Open Ad Hoc Teamwork Experiments

We now describe our experiments, demonstrating how GPL can solve the open AHT problem under full observability. The setup of our open AHT experiments is described in Section 4.9.1. Section 4.9.2 then reports the returns resulting from different versions of GPL and the proposed baselines when solving open AHT problems. Finally, Section 4.9.3 provide a comprehensive analysis of GPL’s joint action-value function after training and discusses why this is the main reason why GPL outperforms the proposed baselines.

4.9.1 Experimental Setup

This section outlines the setup of our open AHT experiments. Section 4.9.1.1 provides an overview of the environments utilised in our experiments. We then describe how we induce openness in Section 4.9.1.2. Section 4.9.1.3 provides a description on the different teammates types utilised for our simulations. Finally, we give an overview of the key features of the different evaluated baselines in Section 4.9.1.4.
4.9. Fully Observable Open Ad Hoc Teamwork Experiments

(a) Level-based Foraging  (b) Wolfpack  (c) FortAttack

Figure 4.3: Environment state visualisations. A visualisation of the state information received by the learner under (a) Level-based Foraging, (b) Wolfpack, and (c) FortAttack, which are the three environments used in our experiments.

4.9.1.1 Environments

Assuming that the environment’s state is always fully observed, we describe three environments for our open AHT experiments:

Level-Based Foraging (LBF). LBF is an environment where the learner must retrieve objects that are positioned across an $8 \times 8$ grid world. The learner, its teammates, and all objects are each assigned a number as their respective level. All agents are then equipped with actions that enable movement along the four cardinal directions and the retrieval of objects positioned in neighbouring grids. An object is retrieved only if the levels of neighbouring agents which chose the retrieve action has a sum that is not less than the object’s own level. After the learner collects an object, the object’s level is given to the learner as a reward.

Wolfpack. In Wolfpack, a learner must collaborate with its teammates to hunt a moving prey inside a $10 \times 10$ grid world. All agents, including the prey, have actions that enable movement across the four cardinal directions. A prey is captured if at least two hunters position themselves adjacent to the prey’s current position on the grid. Given a set of hunters $H$ positioned next to a captured prey, the learner is given a reward of $2|H|$ if it is a member of $H$. Conversely, the learner is given a penalty of $-0.5$ if it is next to a prey without any teammates positioned adjacently to the said prey.

FortAttack. FortAttack is an environment where the learner is part of a defending team that must defend a fort from advancing attackers. The state space in this environment is continuous, and consists of an arena of size $1.8 \times 2$ in which agents can move around. Apart from having actions that enable movement across the four cardinal di-
rections, every agent is equipped with discrete actions that allow them to rotate and shoot opposing team members that venture inside their shooting cones. The episode ends if an attacker reaches the fort, the learner is shot by an attacker, or the attacker fails to reach the fort after a number of 100 timesteps which for each case the learner is given a reward of $-1$, $-0.3$, and $+1$ respectively. Additionally, the learner is given $+0.3$ for successfully shooting an attacker.

4.9.1.2 Environment Openness

In the environments defined in Section 4.9.1.1 we define an upper limit to the number of agents in the environment. This upper limit differs during the training and evaluation stages, which allows us to measure the out-of-distribution generalisation capabilities of the proposed method when dealing with open processes that have never been experienced before. In LBF and Wolfpack, the number of agents is limited to three during training and five during evaluation. On the other hand, there are at most six agents during training and 10 agents during evaluation for FortAttack.

Environment openness is induced differently for the three environments used in our experiments. In LBF and Wolfpack, a teammate only exists in the environment for a certain number of timesteps. If a teammate has existed for longer than its allocated lifetime, it is immediately removed from the environment. A removed teammate is allocated a waiting period, which is the duration before it is pushed into a reentry queue. Given a non-empty reentry queue, agents in the queue re-enter the environment if the number of agents does not exceed the aforementioned upper limit. It is important to note that the reentry queue is randomised, thus inducing stochasticity in how the team composition evolves during learning. For Wolfpack, teammates’ lifetime is sampled uniformly between 25 and 35 timesteps, while the waiting period is sampled uniformly between 15 and 25 timesteps. By contrast, LBF teammates’ lifetime is sampled uniformly between 15 to 25 timesteps, while the waiting period is sampled between 10 and 20 timesteps uniformly.

Unlike LBF and Wolfpack, the changing number of agents in FortAttack directly results from existing agents’ actions. An agent is only removed from the environment if it is shot by a member of the opposing team. After being shot, a shot agent’s distance from the shooter determines the waiting time before it can re-enter the environment. An agent is out for 80 timesteps when its distance to the shooter is the closest possible distance between agents. For other distance values, we use a linear interpolation such that shot agents will have less waiting time the larger their distances are with the
shooter. Finally, at the beginning of the interaction, the number of agents are initialised according to the previously mentioned maximum number of agents, which are divided equally between the attacking and defending team.

### 4.9.1.3 Teammate Types

We create different teammate types to interact with the learner during our open AHT experiments. For all environments, the policy followed by each teammate type is designed by implementing different behavioural heuristics or using MARL-based methods. Policies from different teammate types differ in terms of their efficiency in executing a task, which causes different policies to be needed when interacting with each teammate type. Further details about teammate types utilised for training are available in Appendix [A.1]. Finally, during an interaction, we randomly choose a type from the set of implemented types and assign it to teammates whenever they re-enter the environment.

### 4.9.1.4 Evaluated Algorithms

The algorithms we evaluate in the open AHT experiments can be divided into three categories. Algorithms in the first category implement variations of our proposed GPL method. The second category is a set of single-agent value-based RL algorithms that act as ablations of GPL. Finally, the final category belongs to MARL-based learners. Note that some single-agent and MARL baselines cannot deal with the changing input sizes since they use neural network architectures that receive a fixed-length input. Therefore, we impose a limit on the maximum number of agents allowed in the environment, which allows us to produce fixed-length input vectors for these methods by using placeholder values for features associated with non-existent agents. An overview of these baselines are provided in Table 4.1.

**Graph-based Policy Learning.** We define and evaluate two algorithms based on the GPL method defined in Section 4.1. The first algorithm, GPL-Q, has its joint action value model trained with Q-learning (Watkins and Dayan, 1992). The second algorithm, GPL-SPI, trains the joint action value model with Soft-Policy Iteration (Haarnoja et al., 2018) instead. Aside from this subtle difference in the joint action value model training method, both algorithms use the methods described in Section 4.5 and 4.6 for action selection and learning.

**Single-agent RL baselines.** In alignment with GPL-Q, the single-agent RL base-
Table 4.1: **Single-agent RL baselines.** Comparison between single-agent RL baseline algorithms based on value network architecture alongside the usage of agent & joint action value modelling.

<table>
<thead>
<tr>
<th>Models</th>
<th>GNN</th>
<th>Agent Model</th>
<th>Joint Action-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>QL</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>QL-AM</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GNN</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GNN-AM</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GPL-Q</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPL-SPI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

lines are trained using Q-Learning ([Watkins and Dayan](1992)). These baseline algorithms differ from GPL-Q in terms of the method and model architectures used for action value estimation. At the same time, the single-agent RL baselines also vary in terms of their usage of agent and joint action-value models. While the main characteristics of these baselines are summarised in Table 4.1, details of these baselines alongside the insights obtained by comparing them against GPL-Q and GPL-SPI are provided below:

- **QL.** QL estimates the learner’s action value by directly passing the representations produced by the type inference model into a multilayer perceptron. Comparisons against QL uncover the effects of not using any GNNs in the learner’s model architecture. This comparison also provides insights into direct action value estimation as opposed to using the action value estimation method introduced in Section 4.1.

- **GNN.** The GNN baseline is similar to QL except in its usage of a GNN that uses multi-head attention ([Jiang et al.](2019)) for computing the learner’s action value. Comparing QL’s performance against GNN enables us to identify the gains from using GNNs for action value estimation. At the same time, a comparison between GNN and GPL-Q’s performance enables us to discover the gains from computing the action value following the method described in Section 4.1.

- **QL-AM.** Unlike QL, QL-AM has an additional agent model that predicts teammates’ actions given the type vectors from the type inference model. Each teammate’s predicted action probabilities derived from the agent model are appended...
to their type vectors. The collection of concatenated vectors for every teammate are given as input to the multilayer perceptron to compute the action value of the learner. Comparing QL-AM and QL’s performance helps us understand the gains achieved using an agent model. At the same time, comparing QL-AM and GPL-Q’s performance provides insights into the advantages of using predicted action probabilities through Equation (4.10) instead of using it as input for direct action value estimation.

- **GNN-AM.** GNN-AM is QL-AM with GNN’s multi-head attention-based action value estimation model. The performance comparison between GPL-Q and GNN-AM helps discover the gains resulting from using Equation (4.10) instead of directly utilising predicted action probabilities as input for action value estimation.

**MARL baselines.** We compare the performance of the aforementioned algorithms and MARL-based baselines to demonstrate the deficiencies of MARL methods when solving open AHT. While MARL methods’ utilisation of joint training and their assumption of knowing teammates’ actions prevents it from being a solution for AHT, we can still evaluate the performance of an agent produced by MARL training in open AHT. Our two MARL-based baselines train a group of agents using the MADDPG [Lowe et al., 2017] and DGN [Jiang et al., 2019] respectively. We evaluate these methods in open AHT by sampling an agent from the MARL-based training process and letting it interact with previously unseen teammate types. We choose MADDPG and DGN as our baseline MARL methods since they are both designed for MARL in closed and open environments respectively.

### 4.9.1.5 Training & Evaluation Setup

Following the previously mentioned details of the environment openness and teammate types, we train every algorithm in Section 4.9.1.4 for open AHT. For LBF and Wolfpack, the algorithms are trained for 6.4 million timesteps. On the other hand, these algorithms are trained for 16 million timesteps for FortAttack. At checkpoints which occur every 160000 timesteps the learner’s policy is frozen and evaluated in the training and evaluation task, which only differs in terms of their underlying open process as described in Section 4.9.1.2. This process is repeated across the 8 trained models for each evaluated algorithm, each trained model is initialised with a different random seed. In Section 4.9.2, we report the average perfor-
Figure 4.4: **Open ad hoc teamwork results (training).** Obtained returns for all evaluated environments during training. We show the average and 95% confidence bounds utilising 8 seeds.

The performance in the training task alongside its 95% confidence bounds across 8 runs. The performance reported for any algorithm in the evaluation task is based on the optimal checkpoint with the highest average returns across 8 runs. We choose to report the checkpoint with the highest returns instead of the last checkpoint because of the ad hoc teamwork training objective is to find learner policy as described in Section 3.1.2. Therefore, we argue that the most appropriate checkpoint to be selected for evaluation is the one maximising the returns of the learner during training.

### 4.9.2 Fully Observable Open Ad Hoc Teamwork Results

For every environment described in Section 4.9.1.1, Figure 4.4 shows the training performance of the evaluated algorithms under the open process encountered during training. The result demonstrates that MARL-based methods, such as MADDPG and DGN, consistently achieve worse performance in all environments when compared to other
4.9. Fully Observable Open Ad Hoc Teamwork Experiments

Table 4.2: **Open AHT results (generalisation experiments).** We show the average and 95% confidence bounds during testing utilising 8 seeds. The data was gathered by averaging the returns at the checkpoint which achieved the highest average performance during training. We highlight in bold the algorithm with the highest performance.

<table>
<thead>
<tr>
<th>Env.</th>
<th>GPL-Q</th>
<th>GPL-SPI</th>
<th>QL</th>
<th>QL-AM</th>
<th>GNN</th>
<th>GNN-AM</th>
<th>DDQN</th>
<th>MADDPG</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBF</td>
<td>2.32±0.22</td>
<td>2.40±0.16</td>
<td>1.41±0.14</td>
<td>1.22±0.29</td>
<td>2.07±0.13</td>
<td>1.80±0.11</td>
<td>0.64±0.9</td>
<td>0.91±0.10</td>
</tr>
<tr>
<td>Wolf</td>
<td>36.36±1.71</td>
<td>37.61±1.69</td>
<td>20.57±1.95</td>
<td>14.24±2.65</td>
<td>8.88±1.57</td>
<td>8.07±0.95</td>
<td>2.18±0.96</td>
<td>19.20±2.22</td>
</tr>
<tr>
<td>Fort</td>
<td>14.20±2.42</td>
<td>16.82±1.92</td>
<td>-3.51±0.60</td>
<td>-3.51±1.51</td>
<td>7.01±1.63</td>
<td>8.12±0.74</td>
<td>-5.98±0.82</td>
<td>-4.83±1.24</td>
</tr>
</tbody>
</table>

evaluated algorithms. This is because policies obtained from MARL training are only optimal when interacting with other jointly trained agents. When dealing with previously unseen teammates as in AHT, MARL policies fail to generalise, leading to poor performance.

Figure 4.4 also shows the performance gains resulting from two integral design choices underlying GPL. By comparing the returns from QL, QL-AM, and the other evaluated algorithms, we first discover that GNN-based architectures deliver improved performance by being more suitable for action value estimation under environment openness. Figure 4.4 then highlights the importance of combining joint action value and agent models for estimating the learner’s action value based on Equation 4.10. GPL-Q and GPL-SPI, the only evaluated algorithms utilising Equation 4.10 for action value estimation, have significantly higher returns than the other algorithms. By comparing the performances between GPL-based algorithms alongside single-agent RL baselines without agent models (e.g. QL and GNN) and with agent models (e.g. QL-AM and GNN-AM), we can also conclude that agent models will not improve returns unless they are combined with joint action value estimates for action value estimation. Further analysis of the importance of GPL’s action value estimation method for training will be discussed in Section 4.9.3.

The algorithms’ performance in the evaluation task depicted in Table 4.2 highlights the importance of GNN-based action value estimation to improve generalisation across open processes. Even in LBF, where all but MARL-based baselines deliver similar returns during training, GPL-based methods and GNN-based single-agent RL baselines achieve significantly better generalisation performance than QL and QL-AM. However, GPL-based methods still significantly outperform GNN and GNN-AM in generalisation performance.

The reason GPL outperforms other baselines in terms of generalisation performance lies within its action value estimation method. Single-agent RL baselines yield
Chapter 4. Graph-based Policy Learning

Figure 4.5: **Shooting-related metrics for FortAttack.** We measure the ratio between the number of times a learner successfully shoots an opponent to the number of times it chooses the shoot action, as defined by Equation 4.17. This metric is reported for GPL-Q and the single-agent RL baselines, for each training checkpoint in FortAttack.

significantly worse generalisation performance because their type and action value network do not learn good representations for estimating the learner’s action value in novel open processes. By contrast, GPL estimates the learner’s action value following Equation 4.10. GPL only suffers in environments where the underlying joint action value and teammates’ action distribution do not factorise according to Equation 4.5 and Equation 4.8 as the number of teammates changes.

4.9.3 Joint Action Value Analysis in GPL

In this section, we provide a detailed analysis of the joint action value, which gives insights regarding the higher returns from GPL-Q and GPL-SPI compared to the other baselines in the training tasks. We focus solely on the more complex FortAttack environment for the analysis presented in this section.

We start by analysing the shooting accuracy of the learner which we compute as:

\[
SA(\text{Algorithm}) = \frac{\sum_{(s,a_i,s') \in D_{\text{Algorithm}}} I\{\text{OpponentIsHitByLearner}(s')\} \cdot (\text{True})}{\sum_{(s,a_i,s') \in D_{\text{Algorithm}}} I\{\text{shoot}\} (a_i)}, \tag{4.17}
\]

In the above expression, \( I_A(x) \) denotes the indicator function defined as follows:

\[
I_A(x) = \begin{cases} 
1, & x \in A \\
0, & \text{otherwise}
\end{cases} \quad \tag{4.18}
\]

while \( D_{\text{Algorithm}} \) is defined as a collection of the learner’s state, executed actions, and next states resulting from executing the policy produced by the evaluated algorithm.
Figure 4.6: **Pairwise utility analysis.** (a) We measure $\bar{Q}_{j,k}$, which we define in Equation 4.19 as GPL-Q’s estimate of the contribution towards the learner’s returns resulting from agent $j$ shooting an opponent agent, $k$, under four different scenarios defined for this analysis. Lines 1 and 2 represent $\bar{Q}_{j,k}$ for Scenario 1 and Scenario 2, detailed in Equation 4.20 and 4.21. Lines 3 and 4 represent the value of $\bar{Q}_{j,k}$ in Situation 3 and 4, detailed in Equation 4.22 and 4.23. Surrounding the main plot, we illustrate the four FortAttack interaction scenarios defined for our analysis and visualize an example interaction under each scenario (represented by the white line in black boxes). Each black box is numbered after the scenario that illustrates. Inside the example visualisation of each scenario, the fort is represented by the blue half circle, attackers by red circles, defenders by green circles, the learner with a white dot, and shooting ranges are indicated with dashed view cones. The square matrices near each black box represent the pairwise utility matrix between attackers and defenders connected by the white line, where the yellow marked fields in each square matrix refer to the matrix entries that are averaged over to compute $\bar{Q}_{j,k}$ for the scenarios depicted above and below it.

This reported metric is then computed for each checkpoint of the policies in the training process, with the $D$ Algorithm containing 480000 sample experiences for each algorithm.

Figure 4.5 presents the obtained shooting accuracy results. Based on Figure 4.5, we see that a learner produced via GPL learns to increase its shooting accuracy at a faster rate compared to the baselines. Since shooting is an integral skill for defending the fort, GPL-based learners eventually outperform other learners following its better shooting accuracy. We now analyse various shooting-related metrics and their correlation with the GPL-based learner’s returns to highlight why it outperforms other baselines.

Among the many shooting-related metrics that we evaluated, $\bar{Q}_{j,k}(s)$ is the metric with the highest correlation with a GPL learner’s returns. Given a set of agents $j$ and $k$
alongside the trained pairwise utility estimator, $Q_{S_k}^{i,k}(a_i^t, a_k^t|s_i)$. $\bar{Q}_{j,k}$ is defined as:

$$
\bar{Q}_{j,k}(s) = \frac{\sum_{a^k \in \mathcal{A}_k} Q_{S_k}^{i,k}(a^t = \text{shoot}, a^k|s)}{|\mathcal{A}_k|}.
$$

(4.19)

$\bar{Q}_{j,k}(s)$ is intuitively the estimated pairwise contribution from $j$ towards the learner if $j$ chooses to shoot, which is then averaged across the possible actions of $k$. In FortAttack, note that the learner is always part of the defending team.

We analyse $\bar{Q}_{j,k}(s)$ by first collecting a data set $D$ containing 480000 states, which are obtained by running the learner’s frozen policy at every training checkpoint. $D$ is then used to analyse the average of $\bar{Q}_{j,k}$ under four different scenarios. Assuming $N^{\text{att}}(s)$ and $N^{\text{def}}(s)$ denotes the set of existing agents from the attacking and defending team at state $s$, the reported metrics under the different scenarios are defined below:

- **Scenario 1.** We measure the average $\bar{Q}_{j,k}(s)$ when $k$ is an attacking agent who is inside a defender $j$’s shooting range. Formally, this is defined as:

$$
\bar{Q}_{j,k}^{S_1} = \frac{\sum_{s \in D} \sum_{j \in N^{\text{def}}(s)} \sum_{k \in N^{\text{att}}(s)} \bar{Q}_{j,k}(s) \mathbb{I}_{\{\text{True}\}}(\text{InShootingRange}(j,k))}{\sum_{s \in D} \sum_{j \in N^{\text{def}}(s)} \sum_{k \in N^{\text{att}}(s)} \mathbb{I}_{\{\text{True}\}}(\text{InShootingRange}(j,k))}
$$

(4.20)

- **Scenario 2.** This scenario is the opposite of Scenario 1 where $\bar{Q}_{j,k}(s)$ is averaged for instances when an attacker agent $k$ is not in a defender $j$’s shooting range. This is formally defined as:

$$
\bar{Q}_{j,k}^{S_2} = \frac{\sum_{s \in D} \sum_{j \in N^{\text{def}}(s)} \sum_{k \in N^{\text{att}}(s)} \bar{Q}_{j,k}(s) \mathbb{I}_{\{\text{False}\}}(\text{InShootingRange}(j,k))}{\sum_{s \in D} \sum_{j \in N^{\text{def}}(s)} \sum_{k \in N^{\text{att}}(s)} \mathbb{I}_{\{\text{False}\}}(\text{InShootingRange}(j,k))}
$$

(4.21)

- **Scenario 3.** The average of $\bar{Q}_{j,k}(s)$ is computed assuming that $k$ is a defender within an attacker $j$’s shooting range. The evaluated metric in this scenario is defined as:

$$
\bar{Q}_{j,k}^{S_3} = \frac{\sum_{s \in D} \sum_{j \in N^{\text{att}}(s)} \sum_{k \in N^{\text{def}}(s)} \bar{Q}_{j,k}(s) \mathbb{I}_{\{\text{True}\}}(\text{InShootingRange}(j,k))}{\sum_{s \in D} \sum_{j \in N^{\text{att}}(s)} \sum_{k \in N^{\text{def}}(s)} \mathbb{I}_{\{\text{True}\}}(\text{InShootingRange}(j,k))}
$$

(4.22)

- **Scenario 4.** This scenario is similar to Scenario 3 except $\bar{Q}_{j,k}(s)$ is averaged for instances when defender $k$ is not in attacker $j$’s shooting range. The evaluated metric for this scenario is defined below:

$$
\bar{Q}_{j,k}^{S_4} = \frac{\sum_{s \in D} \sum_{j \in N^{\text{att}}(s)} \sum_{k \in N^{\text{def}}(s)} \bar{Q}_{j,k}(s) \mathbb{I}_{\{\text{False}\}}(\text{InShootingRange}(j,k))}{\sum_{s \in D} \sum_{j \in N^{\text{att}}(s)} \sum_{k \in N^{\text{def}}(s)} \mathbb{I}_{\{\text{False}\}}(\text{InShootingRange}(j,k))}
$$

(4.23)
We now outline important observations regarding the relationship between $\tilde{Q}^{S_1}_{j,k}$ alongside the learner’s returns and shooting accuracy. By comparing $\tilde{Q}^{S_1}_{j,k}$ and the returns of the learner across 100 training checkpoints, we discover that $\tilde{Q}^{S_1}_{j,k}$ and the learner’s returns have a strong positive Pearson correlation coefficient of 0.85. This strong correlation can be seen by comparing the lines associated to $\tilde{Q}^{S_1}_{j,k}$ and to the learner’s returns in Figure 4.6. Comparing Figure 4.5 and Figure 4.6 also shows that a GPL learner starts to become significantly better than baselines in terms of shooting accuracy after $\tilde{Q}^{S_1}_{j,k}$ experiences an uptick in its values, which happens around the 20th checkpoint. These observations highlight the importance of GPL’s pairwise utility estimator ($MLP_\delta$) and more generally its joint action value estimator to achieve high returns in open ad hoc teamwork.

Rather than simply being correlated with the learner’s returns, we highlight GPL’s joint action value model as the main cause behind GPL’s significantly higher returns. The initial increase in value for $\tilde{Q}^{S_1}_{j,k}$ indicates that $MLP_\delta$ starts to see any defender shooting down an attacking team member as advantageous for the learner. Since $MLP_\delta$ is shared between the different agents as mentioned in Section 4.3 and the learner itself is a defender, $MLP_\delta$ also increases the value of the learner shooting down attacking team members. This is an important point as it shows that the learner can derive knowledge directly from its teammates. As learning progresses, we see $MLP_\delta$ further increasing the estimated value of $\tilde{Q}^{S_1}_{j,k}$. This further contrasts the difference between the estimates of $\tilde{Q}^{S_1}_{j,k}$ and $\tilde{Q}^{S_2}_{j,k}$, which drives the learner’s policy to more frequently get attackers inside the learner’s shooting range. These results show that GPL’s joint action value model and its parameter sharing configuration improve the shooting accuracy and the returns of the learner, and it does so by observing teammate behaviour.

Aside from learning to shoot attackers more accurately, GPL’s joint action value model is also responsible for enabling the learner to avoid being shot by attackers. Despite learning this rather late compared to shooting down attackers, Figure 4.6 shows the line associated to $\tilde{Q}^{S_3}_{j,k}$ decreasing as learning progresses. As the value of $\tilde{Q}^{S_3}_{j,k}$ keeps decreasing relative to the value of $\tilde{Q}^{S_4}_{j,k}$, the learner’s policy learns to avoid getting inside any attacker’s shooting range.

We show in the next section that learners resulting from baseline algorithms cannot learn to shoot or evade attackers by observing teammate defenders. Without a joint action value estimation model, a learner can only learn to shoot by experiencing firsthand shooting down attackers. For an initially untrained learner, successfully shooting trained attackers is difficult since getting close to attackers and discovering the proper
orientation alone is challenging to achieve randomly during exploration. Even if a learner manages to get closer to an attacker, their inexperience will likely result in the learner being shot down by the attackers.

### 4.9.4 Action Value Analysis in Single-Agent RL Baselines

![State values for QL.](image1)

(a) State values for QL.

![State values for QL-AM.](image2)

(b) State values for QL-AM.

![State values for GNN.](image3)

(c) State values for GNN.

![State values for GNN-AM.](image4)

(d) State values for GNN-AM.

Figure 4.7: **State values for all single-agent RL baselines.** This visualisation compares the state values in Scenario 1 and 2 for (a) QL, (b) QL-AM, (c) GNN, and (d) GNN-AM. The blue line in each plot shows the average and 95% confidence bounds of $V(s)$ under Scenario 1. By contrast, the green line shows the average and 95% confidence bounds of $V(s)$ under Scenario 2. This figure demonstrates that neither single-agent RL baselines manage to learn the effects of other agents’ actions to the learner.

Following the absence of a joint action value model, this section demonstrates that the single-agent RL baseline algorithms cannot learn the effects of teammates’ actions to improve the learner’s performance. Our analysis follows Section 4.9.3 by being limited to FortAttack. As in the analysis with GPL, we collect a data set of 480000 states at every training checkpoint by running the frozen learner’s policy. We subsequently report measures related to the action value estimates produced by each baseline.
4.9. Fully Observable Open Ad Hoc Teamwork Experiments

While it is impossible to directly compute $\tilde{Q}_{j,k}$ for baseline algorithms due to the absence of CG-based joint action value models, we can use a Monte Carlo estimate to compute state values under specific scenarios. Assuming a set of states that fulfil a specific criterion $S$, the state value under that specific criteria is estimated as:

$$V(S) = \frac{1}{|S|} \sum_{s \in S} V(s), \quad (4.24)$$

with $V(s) = \max_a Q(s, a)$ being the action value of the optimal action at $s$ according to the model produced by the evaluated baseline. We now outline the two evaluation scenarios of interest, defined as the following:

- **Scenario 1.** The first scenario evaluates $V(S_1)$ for a collection of states where an attacker is within the shooting range of any defender.

- **Scenario 2.** The second scenario computes $V(S_2)$ for states where no attacker is within the shooting range of any defender.

These two scenarios correspond to Scenario 1 and 2 in Section 4.9.3 respectively. We limit our analysis in this section to these two scenarios since Section 4.9.3 specifically attributed GPL’s learning performance towards the joint action value model’s ability to evaluate the pairwise utility in these two scenarios. Intuitively, $V(S_1)$ and $V(S_2)$ can be viewed as an approximation of $\tilde{Q}^{S_1}_{j,k}$ and $\tilde{Q}^{S_2}_{j,k}$, defined in Section 4.9.3. By evaluating the difference between $V(S_1)$ and $V(S_2)$, we can determine whether the single-agent RL baselines learn to recognize the value of any defender being in a position to shoot down opposing attackers.

The value of $V(S_1)$ and $V(S_2)$ across the different baselines are reported in Figure 4.7. The results in Figure 4.7 demonstrate that the single-agent RL baselines fail to recognize the advantages of having attackers in the shooting range of any defender. QL and QL-AM instead assign lower average values to states where any defender can shoot down attackers. This negative view of states in Scenario 1 may explain why QL and QL-AM learners have low shooting accuracy and perform poorly during training. On the other hand, GNN and GNN-AM’s average estimate of $V(S_1)$ and $V(S_2)$ also do not highlight the inherent positive difference between the values in Situation 1 and 2, which indicates the failure of both baselines to learn the effects of teammates’ actions to the learner. This inability to understand the effects of others’ actions prevents the baselines from learning important knowledge required to perform well in FortAttack.
Chapter 4. Graph-based Policy Learning

4.10 Chapter Summary

This chapter introduces a method to address the challenging problem of open ad hoc teamwork, in which the goal is to design an autonomous agent capable of robust teamwork under dynamically changing team composition without pre-coordination mechanisms such as joint training. The proposed algorithm GPL uses coordination graphs to learn joint action-value functions that model the effects of other agents’ actions towards the learning agent’s returns, along with a GNN-based RFM model \cite{Tacchetti et al., 2019} trained to predict the actions of other teammates. GPL then combines the output of its joint action-value and agent model to estimate the action-value function the learner utilised to decide its action.

We empirically tested our approach in three multi-agent environments showing that our learned policies can robustly adapt to dynamically changing teams. Furthermore, we also demonstrate that GPL endows a learner with improved generalisation capabilities when deployed in open processes that differ from the one experienced during training. We empirically show that GPL’s success can be attributed to its ability to learn meaningful concepts to explain the effects of other agents’ actions on the learning agent’s returns. This enables GPL to produce action-values that lead to significantly better training and generalisation performances than various baselines.
Chapter 5

Graph-based Policy Learning in Partially Observable Environments

The previous chapter discussed the necessary components to solve the open ad hoc teamwork (AHT) problem in a fully observable setting. In this chapter, we now relax the previous assumptions about full observability and discuss ways to solve the open AHT problem under partial observability. Section 5.1 starts our discussion by highlighting the latent variables that must be inferred to solve an open AHT problem under partial observability. This is followed by a description of an input preprocessing step required for all methods introduced in this thesis in Section 5.2. We then introduce three different neural network models for inferring the previously mentioned latent variables alongside their usage in computing the learner’s optimal action in Sections 5.3, 5.4, and 5.5. The learning objectives and overall pseudocode to train these belief inference models for open AHT are then discussed in Sections 5.6 and 5.7.

We finally outline and discuss the results of the experiments that we conducted in partially observable open AHT problems in Section 5.8. Our experiments evaluate the performance of the different proposed belief inference models when combined with GPL to solve partially observable open AHT problems. We also investigate the information encoded by the belief inference models and their impact on the learner’s performance in partially observable open AHT problems.

5.1 General Overview

In addition to teammates’ types, $\theta_i$, and previous actions, $a_{t-1}^{-i}$, which are not observable by the learner even in fully observable open AHT, partial observability prevents a
Chapter 5. Graph-based Policy Learning in Partially Observable Environments

A learner from observing all teammates’ existence, \( e_t \), and state features, \( s_t \). Following their importance for decision-making in open AHT problems, a learner must infer this collection of unobserved information solely based on its experienced observations. In this section, we define a general framework that a learner may use to infer important latent variables for decision-making in partially observable open AHT problems. These functions are solely based on the learner’s perceived observations and executed actions \( H_t = \{ o_{\leq t}, a^t_{\leq t} \} \). We then define how inferred values of these latent variables are used to estimate the learner’s optimal policy defined in Definition 3.3.

Given \( H_t \), multiple values of inferred latent variables are potentially plausible given \( H_t \). It is therefore helpful to maintain a probabilistic belief over the plausible latent variable values given \( H_t \). As with POMDPs, we call our probabilistic belief over the latent variables given \( H_t \) the belief state. Following the learner-teammates interaction in a PO-OSBG defined in Definition 3.2.1 at each timestep the previous belief state estimate can be updated following the learner’s most recent observation, \( o_t \), and executed action, \( a^t_{t-1} \). Using the Bayes rule, the updated belief state can be evaluated with the following expression:

\[
p(a_{t-1}^i, e_t, s_t, \theta_t | H_t) \propto p(o_t | e_t, s_t) p(e_t, s_t, \theta_t | a_{t-1}^i, e_{t-1}, s_{t-1}, \theta_{t-1}) p(a_{t-1}^i | e_{t-1}, s_{t-1}, \theta_{t-1}) p(a_{t-2}, e_{t-1}, s_{t-1}, \theta_{t-1} | H_{t-1}).
\]

An exact evaluation of \( p(a_{t-1}^i, e_t, s_t, \theta_t | H_t) \) requires knowledge of the distributions appearing in the right-hand side of Equation 5.1 and integrating the right-hand side of Equation 5.1 over the latent variables, which may not have a closed form expression. In such cases, approximate belief updates can be used for estimating Equation 5.1.

In the following sections, we identify three primary ways of approximating beliefs:

1. Maintaining a fixed length vector which contains information about each teammate.

2. Maintaining a particle-based approach that estimates the belief state as a set of particles.

3. Maintaining a distribution of representation vectors that contain information about all latent variables.
Figure 5.1 presents an overview of the three methods introduced in this section. These methods allow the learner to infer the latent information required for solving the open AHT problem under partial observability.

Note that there are two important aspects regarding the training process of our belief inference methods. First, our methods do not assume knowledge over the distributions required for belief update and instead learn them from experience gathered by the learner. Second, we also assume knowledge of the state information during training since recent works have indicated that reconstructing the state information can improve the resulting belief inference methods quality, which translates to a learner’s increased returns at evaluation time (Papoudakis et al., 2021a; Gu et al., 2021).

The remaining step is for the learner to integrate the inferred latent information during action selection. Given a representation that encodes a value of the inferred latent variables \( (e_t, s_t, \theta_t, a_{t-1}^-) \), we can estimate the learner’s optimal action-value function under the latent variables’ value, \( \tilde{Q}_{\pi^*}(e_t, s_t, \theta_t, a^i) \). This is done by combining such a representation with a joint action value module, an agent model module, and an action selection module to compute the learner’s action-value function as introduced in Section 4.1. The resulting representation can be directly used with the other modules for methods that produce a single representation \( \rho \) to infer the latent variables. In contrast, methods that maintain a probabilistic belief over the latent variables must compute the optimal action value function as the expected value of \( \tilde{Q}_{\pi^*}(e_t, s_t, \theta_t, a^i) \) under the belief state following this expression:

\[
\tilde{Q}_{\pi^*}(H_t, a^i) = \int \tilde{Q}_{\pi^*}(e_t, s_t, \theta_t, a^i) p(a_{t-1}^i, e_t, s_t, \theta_t | H_t) \, da_{t-1}^i \, de_t \, ds_t \, d\theta_t.
\]  

Equation 5.2 intuitively expresses \( \tilde{Q}_{\pi^*}(H_t, a^i) \) as the expected value of the state-action value estimate given the belief over the latent variables, all inferred based on the learner experienced observations and actions, \( H_t \).

In the following sections, the definition of each belief approximation model is accompanied by a method that approximates Equation 5.1 to update the belief over latent variables. For clarity, we present each of these methods in different subsections. Section 5.3 presents the fixed length vector representation produced by using autoencoders. Section 5.4 presents particle-based methods. Finally, Section 5.5 presents methods based on variational autoencoders. Note that for each belief inference method’s description, we also detail how to incorporate their inferred latent variable information for decision-making to solve partially observable open AHT problems.
Figure 5.1: **Overview of partially observable methods.** (a) For autoencoder architectures, the observation and actions are encoded into a fixed length vector $\rho_t$. This representation is then sent to a joint action value network to obtain $Q^{\pi^*}_{\rho_t, a_i}$. This value together with the teammates’ actions, as estimated by the decoder network ($q(a_t|\rho_t)$), are used in the action selection module to estimate $\bar{Q}^{\pi^*}_{\rho_t, a_i}$ via marginalisation. (b) Particle-based methods take observation $o_t$ and past actions $a_{t-1}$ and produce a set of particles $U_t$, providing a belief over the latent variables. The particles are then taken as input by a joint action value network, which estimates $Q^{\pi^*}_{\rho_t, a_i}$. The action selection module then combines the output of the joint action value network and the estimated action coming from the action inference module to obtain $\bar{Q}^{\pi^*}_{\rho_t, a_i}$. (c) In variational autoencoder-based belief, we encode the observation and past action to generate parameters to the distribution of representations, $z_t$. Representations from this distribution are then utilised to obtain $\bar{Q}^{\pi^*}_{H_t, a_i}$ by marginalising the output of the joint action value network. Despite their similarity, note that the variational autoencoder and autoencoder-based architectures differ in terms of the output from their encoders.

### 5.2 Input Preprocessing

We outline an input preprocessing method used by all models for solving partially observable open AHT problems proposed in this thesis. Our input preprocessing method for open AHT in partially observable environments assumes knowledge over the set of
agents that can exist in the environment, \( N \). For each agent in \( N \), the input preprocessing method parses the observation into an agent’s visibility in the learner’s observation, \( e_i^t \), and observed features, \( s_i^t \). The visibility and features of teammates not appearing in the learner’s observation are set to a default value of zero. The input preprocessing method also assumes knowledge of the learner’s latest executed action, \( a_{t-1} \).

The input preprocessing method combines the aforementioned information into a collection of feature vectors for belief updates. For each agent \( i \in N \), their observed features and visibility in the observation are concatenated into their combined agent features, \( x_i^t \). The remaining observation features not associated with any agent, \( u \), and \( a_{t-1} \) are subsequently concatenated to each agent’s combined agent features to finally form an observation batch for learning, \( B_{obs} \). This preprocessing step is illustrated in Figure 5.2.

\[ x_i^t = (e_i^t, s_i^t, a_{t-1}) \]

**5.3 Representation-based Latent Variable Inference**

Before we describe approximate inference methods that maintain a belief over the latent variables, this section describes a more straightforward approach that represents all latent variables associated with each teammate \( j \) at timestep \( t \) as a fixed-length vector, \( \rho_j^t \). This approach is inspired by the work of Papoudakis et al. (2021a), which addresses partially observable AHT problems with a fixed number of teammates.

To learn the latent variable representation at timestep \( t \), \( \rho_t \), we utilise an autoen-
coder architecture. The encoder takes $a_{t-1}^i, o_t$, and the latent variable representation from the previous timestep, $\rho_{t-1}$, to compute $\rho_t$. This embedding is then passed to a decoder, which is trained to reconstruct each agent’s existence, state, and actions based on $\rho_t$. Both the encoder and decoder are parameterised by recurrent neural networks described in Section [5.3.1]. We then outline a method to use these representations to select a learner’s optimal action in Section [5.3.2].

5.3.1 Autoencoder Architecture

The autoencoder’s encoder network is implemented as a Long Short-Term Memory (LSTM) network (Hochreiter and Schmidhuber, 1997). At each timestep $t$, the encoder requires the learner’s observation, $o_t$, and previous action, $a_{t-1}^i$, to be processed into a batch of input vectors, $B_{obs}$, following the procedure outlined in Section [5.2]. Given $B_{obs}$, the encoder outputs a representation $\rho_t$ that is defined as the following:

$$\rho_t = \text{MLP}_\alpha(c_t), \quad (5.3)$$

where,

$$c_t, h_t = \text{LSTM}_\alpha(B_{obs}, c_{t-1}, h_{t-1}), \quad (5.4)$$

and $c_{t-1}$ and $h_{t-1}$ are the cell and hidden states of the LSTM at the previous timestep. This encoder is later trained to produce $\rho_t$ that is predictive of the latent variables important for decision-making in partially observable open AHT.

Meanwhile, the decoder network is designed as a neural network that helps the encoder produce $\rho_t$ that is representative of the inferred latent variables. The decoder network receives $\rho_t$ as input and is trained to reconstruct the learner’s current observation alongside predicting the observed teammates’ current actions. The decoder part that reconstructs the learner’s observation is implemented as a multilayer perceptron (MLP), while teammate action prediction is based on GNNs. Given $\rho_t$, the reconstructed observation returned by the decoder is denoted by:

$$B_{pred}(\rho_t) = \text{MLP}_\beta(\rho_t). \quad (5.5)$$

Meanwhile, operations inside the decoder’s component that predicts teammates’ joint actions are defined as follows:

$$q_{\gamma}(a^V_t | \rho_t) = \prod_{j \in \mathcal{V}} p_{\gamma}(a^j | \rho_t), \quad (5.6)$$
assuming $V \subseteq N$ denotes the set of visible teammates and with,

$$
\bar{n}_j = (\text{GNN}_\gamma(\rho_t))_j,
$$

$$
q_\gamma(a^j|\rho_t) = \text{Softmax}(\text{MLP}_\gamma(\bar{n}_j)) (a^j).
$$

\subsection{5.3.2 \textbf{Action Selection}}

Given the fixed-representation $\rho_t$ of the latent variables, we can then utilise GPL to estimate the optimal action-value under partial observability. This optimal action-value function estimation process can be observed in Figure 5.1a. It is important to note that we use the action prediction component of the decoder network as a substitute for GPL’s agent model since it also predicts teammates’ actions given its input representation. Once the value of $\bar{Q}_{\pi^*}(\rho_t, a^i)$ is obtained, the learner chooses the actions that greedily maximise $\bar{Q}_{\pi^*}(\rho_t, a^i)$ at each timestep.

\section{5.4 \textbf{Particle-based Latent Variable Inference}}

This section provides our first proposed approach for maintaining a belief over important latent variables for solving partially observable AHT problems. This approach represents the belief over inferred latent variables as a collection of sampled particles from $p(a_{t-1}^i, e_t, s_t, \theta_t|H_t)$. These particles will be input for GPL to decide the learner’s optimal action under partial observability.

Section 5.4.1 starts our discussion by outlining how sampled particles are represented in a graph-based representation suitable as input for GPL. Section 5.4.2 outlines a method to update the belief representation using neural network models which receive the learner’s most recent observation and action as input. Further details on the architectures of the neural network models involved in belief update are provided in Appendix 5.4.3. Then, in Section 5.4.4 we outline a method to select optimal actions based on the particle representation.

\subsection{5.4.1 \textbf{Belief Representation}}

A particle-based belief representation estimates belief over latent variables at time $T$ as a collection of particles, which is denoted by $U_t$, sampled from the distribution of latent variables. There are two motivations for representing belief as a collection of particles. First, it provides the flexibility to estimate belief states that do not belong
to any particular family of distributions. Second, it enables a tractable optimal policy
computation.

Previous works have utilised particle-based representations for solving single-agent RL problems under partial observability (Igl et al., 2018; Singh et al., 2021). However, these works have not been extended to open AHT problems where it is necessary to maintain a belief not only over the environment’s state, but also over the existence, types, and actions of agents in the environment. In addition, a suitable belief representation for open AHT also needs to account for the environment’s openness.

We extend the particle-based approach for solving partial observability in open AHT by defining a particle \( u_k \in U_t \) as a 5-tuple \( < a_{t-1}^{u_k}, e_t^{u_k}, s_t^{u_k}, \theta_t^{u_k}, w_t^{u_k} > \). We assume knowledge over the set of all agents \( N \) that can exist in a PO-OSBG and encode a possible value of all latent variables associated with each agent, \( i \in N \), in a particle. For a particle \( u_k \), each of its components is defined as follows:

- \( a_{t-1}^{u_k} \in A_N \) is the joint action of all agents in \( N \) at the previous timestep.
- \( s_t^{u_k} \in \mathbb{R}^{|N| \times m} \) is a collection of vectors of length \( m \), which represents the inferred state feature of each agent in \( N \).
- \( e_t^{u_k} \in \{0, 1\}^{|N|} \) are indicator variables indicating the existence of each agent in \( N \).
- \( \theta_t^{u_k} \in \mathbb{R}^{|N| \times n} \) are vectors that denote the inferred types of each agent in \( N \).
- \( w_t^{u_k} \in \mathbb{R} \) is the log likelihood of \( < a_{t-1}^{u_k}, e_t^{u_k}, s_t^{u_k}, \theta_t^{u_k} > \) given \( H_t \).

Note that our particle representation represents agents’ states and types as vectors of length \( m \) and \( n \) respectively since we assume no knowledge regarding the underlying state and type space of the PO-OSBG. Furthermore, the state, type, and joint actions associated to agents that are deemed non-existent are set to default values of 0. Given \( U_t \), the belief state at \( t \) is then estimated as:

\[
p(a_{t-1}^{-i}, e_t, s_t, \theta_t | H_t) = \sum_{u_k \in U_t} \left( \frac{1_{\{ < a_{t-1}^{u_k}, e_t^{u_k}, s_t^{u_k}, \theta_t^{u_k} > \}} \langle a_{t-1}^{-i}, e_t, s_t, \theta_t \rangle \exp(w_t^{u_k})}{\sum_{u_j \in U_t} \exp(w_t^{u_j})} \right),
\]

(5.8)

with \( 1_A(x) \) denoting the indicator function defined in Equation 4.18.

### 5.4.2 Belief Update

The particle-based belief representation is updated by applying the AESMC technique (Le et al., 2018), which is an approximate inference technique to update particle-
5.4. Particle-based Latent Variable Inference

Given the learner’s most recent observation, $o_t$, and executed action, $a^t_{t-1}$, at timestep $t$ our proposed approach approximates the distribution over all agents’ existence, $e_t$, feature representations, $s_t$, types, $\theta_t$, and past joint actions, $a_{t-1}$, as a collection of graph-based particles produced by the latent variable inference network. The learner’s belief is updated at each timestep by recomputing the contents of each particle from $t - 1$ through a sequential execution of the following steps: (i) sampling previous particles based on their log weights, $w_{t-1}$, (ii) a prediction step, which consists of action inference, state inference and type update (iii) a particle weight update step. The sampling operations (and deterministic updates in the case of type and particle weight update) produce updated contents $(a_{t-1}, (e_t, s_t), \theta_t, w_t)$ for the sampled particles, which specific measures they represent are outlined in Section 5.4.1.

Based latent variable estimates in stochastic processes such as PO-OSBGs. This update utilises a collection of distributions, which perform stochastic updates to the latent variable estimates from each particle based on $o_t$ and $a^t_{t-1}$. In its application to a PO-OSBG, the log-likelihood of each particle is recomputed by AESMC based on the updated latent variable estimates’ likelihood according to an estimate of the right-hand side of Equation 5.1. An illustration of the AESMC-based update in inferring latent

Figure 5.3: Overview of particle-based belief update. Given the learner’s most recent observation, $o_t$, and executed action, $a^t_{t-1}$, at timestep $t$ our proposed approach approximates the distribution over all agents’ existence, $e_t$, feature representations, $s_t$, types, $\theta_t$, and past joint actions, $a_{t-1}$, as a collection of graph-based particles produced by the latent variable inference network. The learner’s belief is updated at each timestep by recomputing the contents of each particle from $t - 1$ through a sequential execution of the following steps: (i) sampling previous particles based on their log weights, $w_{t-1}$, (ii) a prediction step, which consists of action inference, state inference and type update (iii) a particle weight update step. The sampling operations (and deterministic updates in the case of type and particle weight update) produce updated contents $(a_{t-1}, (e_t, s_t), \theta_t, w_t)$ for the sampled particles, which specific measures they represent are outlined in Section 5.4.1.
variables in a PO-OSBG is provided in Figure 5.3.

The models used for updating the particle-based belief estimate are grouped in the latent variable inference network, which approximates the update in Equation 5.1 following three steps: i) particle sampling, ii) prediction step, and iii) particle log-likelihood update. The details of the updates in each step are provided below:

**Particle Sampling.** Given \( w_{t-1} = \{ w_{t-1}^u | u \in U_{t-1} \} \), the first step is to sample particles from \( U_{t-1} \) with replacement based on their log likelihood:

\[
u_1, u_2, \ldots, u_K \overset{\text{iid}}{\sim} \text{Categorical}(\text{Softmax}(w_{t-1})).\quad (5.9)
\]

We denote the collection of \( K \) sampled particles as \( \bar{U}_{t-1} \). For each \( u_k \in \bar{U}_{t-1} \), the contents of \( u_k \) are updated in the subsequent steps.

**Prediction Step.** The prediction step updates the estimated values of the state, action, existence and type of each agent in every particle \( u_k \in \bar{U}_{t-1} \) at time \( t \). This sequential update starts with the action, as seen in Figure 5.3. The action update is followed by a process that updates agents’ state representations and existence. Finally, the type representation of each agent is updated. For each particle component, we utilise proposal distributions to incorporate important information not originally involved in the belief update process outlined in Equation 5.1. Specifically, we incorporate the learner’s observation \( o_t \) and most recent action \( a_{i-1} \) when updating the particle representation.

To update the joint action component of each particle, given \( o_t \) and \( a_{i-1} \), we introduce a proposal action distribution, \( q_{\alpha p}(a_{t-1}^u | e_{t-1}^u, s_{t-1}^u, \theta_{t-1}^u, a_{i-1}, o_t) \). For each particle \( u_k \in \bar{U}_{t-1} \), we draw a sample from the proposal distribution such that,

\[
a_{t-1}^u \sim q_{\alpha p}(a_{t-1}^u | e_{t-1}^u, s_{t-1}^u, \theta_{t-1}^u, a_{i-1}, o_t),
\]

and use \( a_{t-1}^u \) as the updated joint action of each particle \( u_k \). In \( a_{t-1}^u \), note that actions of teammates deemed to not have existed in the previous timestep by \( u_k \) are set to a default value of no action. Furthermore, the learner’s known previous action is set to its observed value \( a_{i-1} \).

After updating the joint actions, \( e_{t-1}^u \) and \( s_{t-1}^u \) are updated according to \( o_t, a_{t-1}^u \), and the newly updated value of \( a_{t-1}^u \) for all \( u_k \in \bar{U}_{t-1} \). This update is based on sampling from the updated teammate existence and state representation from the proposal state distribution such that:

\[
e_{t-1}^u, s_{t-1}^u \sim p_{\beta}(e_{t-1}^u, s_{t-1}^u | e_{t-1}^u, s_{t-1}^u, a_{t-1}^u, a_{t-1}^i, o_t).
\]


Like the proposal action distribution for joint action inference, we sample from the proposal distribution to account for \( o_t \) and \( a'_{t-1} \) in updating \( e^{uk}_t \) and \( s^{uk}_t \). It is also important to note that the state is updated based on the predicted existence following \( s^{uk}_t = e^{uk}_t \cdot \delta_{t} \).

The next step is to update the inferred teammate types \( \theta^{uk}_t \) for each particle in \( \bar{U}_{t-1} \). Teammate types are updated based on the sampled \( a^{uk}_{t-1}, e^{uk}_{t-1} \) and \( s^{uk}_{t-1} \). While teammates deemed non-existent (\( e^{uk}_{t-1} = 0 \)) are assigned a type vector of \( \mathbf{0} \), existing agents’ types undergo a deterministic update using the \textit{type update network} parameterised by \( \delta \) following this expression:

\[
\theta^{uk}_t = f_{\delta}(s^{uk}_t, \theta^{uk}_{t-1}, a^{uk}_{t-1}) \tag{5.12}
\]

**Particle Weight Update.** The final step in the particle-based belief update is to update the log-likelihood of particles in \( \bar{U}_{t-1} \). Note that particles’ log-likelihood cannot be updated based on the aforementioned proposal distributions alone. Specifically, the approximated belief update in Equation [5.1] is defined over target distributions that are not conditioned on the learner’s most recent observation. To compensate for how particle values are updated based on proposal distributions that are different from the estimated target distributions, we apply importance sampling correction when updating the weights of each particle.

After sampling \( a^{uk}_{t-1} \), the likelihood of this sampled joint action is incorporated when updating the log-likelihood of the new set of particles in \( \bar{U}_{t-1} \). The likelihood of \( a^{uk}_{t-1} \) is evaluated based on the target action distribution, \( q_{\alpha}(a^{uk}_{t-1} | e^{uk}_{t-1}, s^{uk}_{t-1}, \theta^{uk}_{t-1}) \), which is used to update the belief in Equation [5.1]. Since we sample from a different distribution to incorporate \( o_t \) and \( a'_{t-1} \) to update \( a_{t-2} \), additional corrections are done to the likelihood computation, which results in the following joint action likelihood expression:

\[
w^{uk}_{t-1, \alpha} = \log \left( \frac{q_{\alpha}(a^{uk}_{t-1} | e^{uk}_{t-1}, s^{uk}_{t-1}, \theta^{uk}_{t-1})}{q_{\alpha}(a^{uk}_{t-1} | e^{uk}_{t-1}, s^{uk}_{t-1}, \theta^{uk}_{t-1}, a_{t-1})} \right). \tag{5.13}
\]

The sampled \( e^{uk}_{t} \) and \( s^{uk}_{t} \) is also utilised for updating the log likelihood of each particle in \( \bar{U}_{t-1} \). We specifically compute the likelihood of \( e^{uk}_{t} \) and \( s^{uk}_{t} \) according to the target state distribution, \( p_{\beta}(s^{uk}_{t} | e^{uk}_{t}, a^{uk}_{t-1}) \), which is used for the Bayesian belief update in Equation [5.1]. To account for sampling \( e^{uk}_{t} \) and \( s^{uk}_{t} \) from the proposal distribution, the likelihood of \( s^{uk}_{t} \) under both distribution is then evaluated following this expression:

\[
w^{uk}_{t-1, \beta} = \log \left( \frac{p_{\beta}(s^{uk}_{t} | e^{uk}_{t}, a^{uk}_{t-1})}{p_{\beta}(s^{uk}_{t} | e^{uk}_{t-1}, s^{uk}_{t-1}, a^{uk}_{t-1})} \right). \tag{5.14}
\]
An additional term is taken into consideration for updating the particle weight, which is based on the observations $o_t$, and is defined as:

$$ w_{t-1,\zeta}^u = \log(p_{\zeta}(o_t|s_{t}^u, a_{t-1}^u)), \quad (5.15) $$

with $p_{\zeta}(o_t|s_{t}^u, a_{t-1}^u)$ being the observation likelihood distribution, which evaluates the likelihood of a learner’s observation given $s_{t}^u$ and $a_{t-1}^u$ resulting from the state and joint action inference step during the update.

Approximating the belief update provided by Equation [5.1], a sampled particle’s log-likelihood is updated following this expression:

$$ w_{t}^u = w_{t-1,\zeta}^u + w_{t-1,\beta}^u + w_{t-1,\alpha}^u. \quad (5.16) $$

The updated content of each particle $u_k \in \hat{U}_{t-1}$ is then used as an estimate of the current belief state, $U_t = \{(a_{t}^u, s_{t}^u, \theta_{t}^u, w_{t}^u)|u_k \in \hat{U}_t\}$. Note that Equation [5.16] estimates Equation [5.1] while accounting for the usage of samples generated from proposal distributions that incorporate $o_t$ and $a_{t-1}^i$ to update the particles. Finally, $w_{t-1}$ is not considered in Equation [5.16] since the particle sampling step implicitly accounts for the particles’ weights from the previous timestep.

### 5.4.3 Model Architecture

The distributions and functions responsible for updating the particle-based belief estimate are implemented as neural network models. The details of these neural network models are as follows:

**Joint Action Inference Models.** The proposal and target action distribution for updating the joint action component in a particle are implemented as networks with similar architecture. The only difference is that the proposal action distribution includes $o_t$ and $a_{t-1}^i$ as its input, assuming $o_t$ and $a_{t-1}^i$ are preprocessed into $B_{obs}$ following the method outlined in Section [5.2]. The input for the neural network representing the proposal and target distribution is defined as follows:

$$ D_{in} = \begin{cases} \text{Concatenate}(e_{t-1}^u, s_{t-1}^u, \theta_{t-1}^u), & \text{if target distribution} \\ \text{Concatenate}(e_{t-1}^u, s_{t-1}^u, \theta_{t-1}^u, B_{obs}, a_{t-1}^i), & \text{otherwise}. \end{cases} \quad (5.17) $$

The network architecture to compute the proposal and target action distribution then evaluates the joint action probability distribution as:
5.4. Particle-based Latent Variable Inference

\[ q_\alpha(a_t|D_{\text{in}}) = \prod_{j \in N} q_\alpha(a_j|D_{\text{in}}), \]  
(5.18)

with,

\[ \bar{n}_j = (\text{GNN}_\alpha(D_{\text{in}}))_j, \]
\[ q_\alpha(a_j|D_{\text{in}}) = \text{Softmax}(\text{MLP}_\alpha(\bar{n}_j))(a_j). \]
(5.19)

With \( \alpha^p \) and \( \alpha^t \) respectively being the parameters of the proposal and target action distribution, our implementation uses separate neural networks for estimating these distributions, such that \( \alpha = (\alpha^p, \alpha^t) \).

**Existence and State Inference Models.** As in the case with the joint action inference models, the input for models estimating the proposal and target distributions during existence and state inference is derived from concatenating all the necessary information as defined below:

\[ D_{\text{in}} = \begin{cases} 
\text{Concatenate}(e_{i-1}^{u_k}, s_{i-1}^{u_k}, a_{i-1}^{u_k}), & \text{if target distribution} \\
\text{Concatenate}(e_{i-1}^{u_k}, s_{i-1}^{u_k}, a_{i-1}^{u_k}, B_{\text{obs}}, a_{i-1}^{i}), & \text{otherwise}. 
\end{cases} \]
(5.20)

For existence inference, we assume a unique integer index assigned to each agent in \( N \), where the index assigned to \( j \in N \) is denoted as \( j_{\text{id}} \). Denoting \( e_{\text{t}}^{<j_{\text{id}}} \) and \( D_{\text{in}}^{<j_{\text{id}}} \) as the set of inferred existence and state features of teammates that are assigned to a smaller index than \( j \), both the target and proposal distribution are implemented as a neural network which computes agents’ existence in the following manner:

\[ p_\beta(e_{i}^{u_k}|D_{\text{in}}) = \prod_{j \in N} p_\beta(e_{i}^{u_k, j}|e_{\text{t}}^{<j_{\text{id}}}, D_{\text{in}}^{<j_{\text{id}}}), \]
(5.21)

with,

\[ \bar{n}(j) = \sum_{\{k|k_{\text{id}}<j_{\text{id}}\}} \text{MLP}_\alpha(\text{Concatenate}(e_{i}^{k}, D_{\text{in}}^{k})), \]
\[ E_{\text{in}}^j = \text{Concatenate}(e_{i-1}^{j}, D_{\text{in}}^{j}, \bar{n}(j)), \]
\[ p_\beta(e_{i}^{u_k, j} = 1|e_{\text{t}}^{<j_{\text{id}}}, D_{\text{in}}^{<j_{\text{id}}}) = \text{Sigmoid}(\text{MLP}_\alpha(E_{\text{in}}^j)). \]
(5.22)

This autoregressive existence inference technique resembles GraphRNN (You et al., 2018), which is a generative model that generates graphs with varying numbers of nodes in an autoregressive fashion.

For state inference, both the proposal and target distributions are represented as a multivariate normal distribution with a diagonal covariance matrix, \( \mathcal{N}((\mu_\beta, \Sigma_\beta)) \), which
parameters are evaluated by neural networks following this expression:

\[
\mu_\beta(D_{in}) = \text{MLP}_\mu^\beta(D_{in}),
\]

\[
\Sigma_\beta(D_{in}) = \text{Softplus}(\text{MLP}_\Sigma^\beta(D_{in})),
\]

In our implementation, the target and proposal distribution for teammate existence and state inference is implemented as separate models which parameters denoted as $\beta^t$ and $\beta^p$ respectively.

**Type Update Network.** We implement the type update network as an LSTM which accounts for agents’ previous types and recently inferred state representation and actions to compute their respective types. The type update process in the type update network is provided in the following expression:

\[
c_{t-1}^{\mu_k}, h_{t-1}^{\mu_k} = \text{LSTM}_{\delta}(D_{in}, c_{t-1}^{\mu_k}, h_{t-1}^{\mu_k}),
\]

\[
\theta_{t}^{\mu_k} = \text{MLP}_{\delta}(c_{t}^{\mu_k}),
\]

with $c_{t-1}^{\mu_k}$ and $h_{t-1}^{\mu_k}$ being the cell and hidden state that represents the sequence of previously inferred state and type representations. The input to the LSTM model is then defined as:

\[
D_{in} = \text{Concatenate}(s_{t}^{\mu_k}, \theta_{t-1}^{\mu_k}, a_{t-1}^{\mu_k}).
\]

**Observation Likelihood Model.** We assume that the observation vector we reconstruct is the preprocessed data vector, $B_{obs}$. Since $B_{obs}$ is a collection of continuous vectors, we use a multivariate normal distribution, $\mathcal{N}(\mu_\zeta, \Sigma_\zeta)$, which parameters are computed as defined below:

\[
\mu_\zeta(D_{in}) = \text{MLP}_\mu^\zeta(D_{in}),
\]

\[
\Sigma_\zeta(D_{in}) = \text{Softplus}(\text{MLP}_\Sigma^\zeta(D_{in})),
\]

with the input to this model defined as:

\[
D_{in} = \text{Concatenate}(s_{t}^{\mu_k}, c_{t-1}^{\mu_k}).
\]

**5.4.4 Action Selection**

Representing our belief estimates as a collection of particles enables us to circumvent a challenge in evaluating the learner’s optimal action-value function. As mentioned in Section 5.1, $p(a_{t-1}, e_t, s_t, \theta_t|H_t)$ can be combined with the different modules of GPL to compute the learner’s optimal action-value function under partial observability. A
5.4. Particle-based Latent Variable Inference

Figure 5.4: **Overview of particle-based action selection.** Given the updated set of particles ($U_t$) at time $t$, the Joint Action Value Network utilises this representation to provide a particle-based approximation of $Q_{\pi_i}^*(e_{u_k}^{t}, s_{u_k}^{t}, \theta_{u_k}^{t}, a^i)$. The Action Selection Module carries a two-step process. First, it marginalises over $Q_{\pi_i}^*(e_{u_k}^{t}, s_{u_k}^{t}, \theta_{u_k}^{t}, a^i)$, with the teammate action probability $q_\alpha(a_t-1|U_t)$ coming from the action inference module in the Latent Variable Inference Network. Second, the resulting particle-based state value $\tilde{Q}_{\pi_i}^*(H_t, a^i)$ is then collapsed to a single representation based on the particle weight $w_t$ to obtain $\tilde{Q}_{\pi_i}^*(H_t, a^i)$, following Equation 5.31.

A problem arises for exact evaluation of Equation 5.8 when $\tilde{Q}_{\pi_i}^*(e_t, s_t, \theta_t, a^i)$ is implemented as a neural network since the integral generally does not have a closed-form expression.

By using a particle-based belief representation, we avoid integrating over all possible values of the latent variables. This process is detailed in Figure 5.4. Substituting Equation 5.8 into $p(a_{t-1}, e_t, s_t, \theta_t|H_t)$ in Equation 5.2 results in the following expression:

$$\tilde{Q}_{\pi_i}^*(H_t, a^i) = \sum_{u_k \in U_t} \left( \frac{\exp(w_{u_k}^t)}{\sum_{u_j \in U_t} \exp(w_{u_j}^t)} \right) \tilde{Q}_{\pi_i}^*(e_{u_k}^t, s_{u_k}^t, \theta_{u_k}^t, a^i),$$

which is only a summation of functions defined over the contents of the particles. In the above expression, we estimate $\tilde{Q}_{\pi_i}^*(e_t, s_t, \theta_t, a^i)$ like GPL by marginalising over the output of the joint action value model, $Q_{\pi_i}^*(e_{u_k}^{t}, s_{u_k}^{t}, \theta_{u_k}^{t}, a^i)$, as seen in Figure 5.4. The learner then greedily chooses actions that maximise $\tilde{Q}_{\pi_i}^*(H_t, a^i)$ at any timestep.
5.5 Variational Autoencoder-based Latent Variable Inference

A problem occurs under particle-based approaches as more particles are required to estimate a distribution when the dimension of inferred latent variables increases or if distributions required for updating the particle contents have high variance (Murphy and Russell, 2001). Ensuring an accurate representation of the belief posterior with many particles is computationally expensive to maintain and update. In this section, we provide an alternative method that does not maintain a collection of particles representing belief over important latent variables for open ad hoc teamwork. Instead, all latent variables are represented as a single representation vector whose distribution is assumed to be a parametric distribution.

5.5.1 Belief Representation & Update

The alternative approach is to instead represent belief as a distribution of representation vectors, \( z_t \in \mathbb{R}^{N \times m} \). The belief over \( z_t \), \( p(z_t | H_t) \), is then evaluated given the learner’s interaction experience \( H_t \). We prevent maintaining a large collection of particles by ensuring this distribution is a parametric distribution with low variance, which parameters are estimated by a trained model that receives \( H_t \) as input. The model is trained to ensure that higher likelihood is associated with sampling representations \( z_t \) that are more informative of the interaction experience \( H_t \). Sampled values of \( z_t \) then provide relevant information for action value computation.

We achieve our goal of training a model for estimating \( p(z_t | H_t) \) using variational autoencoders (VAEs) (Kingma and Welling, 2013). To ensure \( z_t \) is informative of \( H_t \), VAEs assume the existence of an underlying generative model, \( p(H_t | z_t) \), that determines the way \( H_t \) is generated from \( z_t \). Given a prior distribution on \( z_t \), the true posterior over \( z_t \), \( p(z_t | H_t) \), may then be evaluated via the Bayes theorem:

\[
p(z_t | H_t) = \frac{p(H_t | z_t)p(z_t)}{\int_{z_t} p(H_t | z_t)p(z_t)dz_t}.
\]  

(5.32)

An exact evaluation of Equation (5.32) is generally intractable since the integral operation does not have a closed-form expression. Instead, VAEs estimate the posterior with a variational parametric distribution, \( q(z_t | H_t) = \mathcal{N}(z_t; \mu, \Sigma) \). The variational parametric distribution is optimised to minimise the Kullback-Liebler divergence between the two distributions, \( D_{KL}(q(z_t | H_t) || p(z_t | H_t)) \).
Both \( p(H_t|z_t) \) and \( q(z_t|H_t) \) are represented by VAEs as parametric distributions which parameters are estimated by neural network models called the decoder and encoder respectively. At each timestep, updates to the learner’s belief over the latent variables are done by computing the distribution parameters of \( q(z_t|H_t) \) based on \( H_{t-1} \) and the learner’s most recent observation and action. Details of the network architectures that we use to represent the encoder and decoder are provided in Section 5.5.2. The objective functions for training the VAE’s encoder and decoder are then provided in Section 5.6.2.

### 5.5.2 Model Architecture

We now detail the model architecture of the variational autoencoder used in this proposed approach. This network receives a batch of vectors, \( B_{obs} \), obtained by preprocessing \( o_t \) and \( a_{t-1} \) based on the preprocessing method outlined in Section 5.2. At every timestep, \( B_{obs} \) is used as input to the encoder architecture to compute the distribution over latent variables \( z_t \). Furthermore, \( B_{obs} \) also acts as the information which will be reconstructed by the decoder. The architecture of the encoder and decoder is provided below:

**Encoder Network.** The encoder network is implemented as an LSTM which receives the learner’s preprocessed observation, \( B_{obs} \), as input. It then produces the mean and covariance matrix for the variational parametric distribution following this expression:

\[
\mu_t = \text{MLP}_\omega(c_t), \tag{5.33}
\]

\[
\Sigma_t = \text{MLP}_\omega(c_t), \tag{5.34}
\]

where,

\[
c_t, h_t = \text{LSTM}_\alpha(B_{obs}, c_{t-1}, h_{t-1}), \tag{5.35}
\]

with \( c_{t-1} \) and \( h_{t-1} \) being the LSTM’s cell and hidden state that represents the sequence of previous observations.

**Decoder Network.** The decoder is trained to reconstruct the learner’s observations alongside predicting its observed teammates’ actions since its role is to reconstruct the information observed by the learner. The decoder network then outputs both the likelihood of \( B_{obs} \) alongside the likelihood of observed teammates’ actions. Since \( B_{obs} \) is a collection of continuous vectors, we compute the likelihood of \( B_{obs} \) based
on a multivariate normal distribution, \( \mathcal{N}(\mu_\beta, \Sigma_\beta) \), which parameters are computed as defined below:

\[
\begin{align*}
\mu_\beta(z_t) &= \text{MLP}_\mu^\beta(z_t), \\
\Sigma_\beta(z_t) &= \text{Softplus}(\text{MLP}_\Sigma^\beta(z_t)),
\end{align*}
\]

assuming \( z_t \) is sampled from the variational parametric distribution outputted by the learner.

On the other hand, the part of the decoder that predicts the likelihood of teammates’ actions has a similar implementation as the joint action inference model for the particle-based approach in Section 5.4.3. Given \( z_t \), the decoder computes the likelihood of observed agents’ actions using a GNN following this equation:

\[
p_\gamma(a^V_t|z_t) = \prod_{j \in V} p_\gamma(a_j|z_t),
\]

assuming \( V \subseteq N \) denotes the set of visible teammates and with,

\[
\bar{n}_j = (\text{GNN}_\gamma(z_t))_j, \\
p_\gamma(a^j_t|z_t) = \text{Softmax}(\text{MLP}_\gamma(\bar{n}_j))(a^j).
\]

\[
\text{5.5.3 Action Selection}
\]

Given the variational parametric distribution, the action value under partial observability is computed as:

\[
\bar{Q}_{\pi_i}^*(H_t, a^i) = \int q(z_t|H_t) \bar{Q}_{\pi_i}^*(z_t, a^i)dz_t.
\]

\( \bar{Q}_{\pi_i}^*(z_t, a^i) \) denotes the action value estimate based on Equation 4.10 given \( z_t \) as input. However, exact evaluation of Equation 5.40 is not possible since the integral generally does not have a closed form expression when \( \bar{Q}_{\pi_i}^*(z_t, a^i) \) is represented as a neural network.

To approximate Equation 5.40, we instead adopt a Monte Carlo approach. We sample \( n \) samples from \( q(z_t|H_t) \),

\[
z_1^t, z_2^t, \ldots, z_n^t \sim q(z_t|H_t),
\]

and estimate \( \bar{Q}_{\pi_i}^*(H_t, a^i) \) based on the following Equation:

\[
\bar{Q}_{\pi_i}^*(H_t, a^i) = \frac{\sum_{k=1}^n \bar{Q}_{\pi_i}^*(z^k_t, a^i)q(z^k_t|H_t)}{\sum_{l=1}^n q(z^l_t|H_t)}.
\]
5.6 Learning Objective

The aforementioned latent variable inference models are trained alongside GPL to infer important latent information for decision-making and use it for action selection in partially observable open AHT. During execution, the learner has only access to its observations and past actions. However, as it helps improve the performance of the methods, we assume that the learner also has access to the environment state and the observed teammates’ joint actions to train its models during training. Knowing the full state of the system during training is a common assumption in partially observable environments (Gu et al., 2021; Papoudakis et al., 2021a) that helps provide learners with higher returns. Therefore, given a set of interaction experiences,\n
\[ D = \{ \{ (s^n_t, o^n_t, V^n, r^n_t, o^n_{t+1}) \}_{t=1}^{T_n} \}_{n=1}^{D} \],

we train the models on the following loss function:

\[
L_{\text{Pinf}, Pst, P_{ag}, P_{val}}(D) = L_{\text{INF}}^{\text{Pinf}}(D) + L_{\text{SR}}^{\text{Pinf}, \cup P_{st}}(D) + L_{\text{NLL}}^{\text{P_{ag}}}(D) + L_{\text{RL}}^{\text{P_{val}}}(D). \tag{5.43}
\]

In the above equation, \( P_{\text{inf}}, P_{\text{st}}, P_{\text{ag}} \) and \( P_{\text{val}} \) denote the collection of model parameters for latent variable inference, state reconstruction, GPL’s agent model and joint action value models respectively.

While its computation may differ across the latent variable inference model being used, each of the terms on the right-hand side of Equation 5.43 fulfil an important role in the optimisation process. \( L_{\text{INF}}^{\text{Pinf}}(D) \) serves as the loss function that is optimised by the latent variable inference models to produce representations for decision-making. \( L_{\text{SR}}^{\text{Pinf}, \cup P_{st}}(D) \) is the state reconstruction loss, which aligns with previous work that uses privileged state information to train the belief inference model to produce representations that are more informative of the state (Papoudakis et al., 2021a). On the other hand, \( L_{\text{NLL}}^{\text{P_{ag}}}(D) \) and \( L_{\text{RL}}^{\text{P_{val}}}(D) \) are the negative log-likelihood and value-based RL losses which we introduce in Section 4.6 to train GPL for solving open ad hoc teamwork.

We provide details of regarding the computation of \( L_{\text{INF}}^{\text{Pinf}}(D), L_{\text{SR}}^{\text{Pinf}, \cup P_{st}}(D), L_{\text{NLL}}^{\text{P_{ag}}}(D), L_{\text{RL}}^{\text{P_{val}}}(D) \) across the previously defined belief inference models in the following sections.

5.6.1 Particle-based Belief Inference Models

Belief Inference Loss Function. In the model introduced in Section 5.4, the negative ELBO loss is defined as a function of the belief model parameters, \( P_{\text{inf}} = (\alpha, \beta, \delta, \zeta) \). Following AESMC (Le et al., 2018), \( P_{\text{inf}} \) is trained to minimize the negative ELBO
defined as:

\[
L_{\text{ELBO}}^{\text{inj}}(D) = - \sum_{H_n \in D} \log \left( \sum_{u_k \in U_n} \exp(w_{T_n}^{u_k}) \right),
\]

with \( U_n \) being the collection of particles resulting from applying the belief inference procedure in Section 5.4.2 to \( H_n \).

**State Reconstruction Loss Function.** The state reconstruction loss is computed based on the set of particles, \( U_n \), computed in Equation 5.45. Given \( U_n \) and a state reconstruction distribution parameterised by \( P_{\text{st}} = \{\theta\} \), the state reconstruction loss function is defined as:

\[
L_{\text{SR}}^{P_{\text{inj}}(H_n)}(D) = - \sum_{H_n \in D} \sum_{u_k \in U_n} \log(p_{\theta}(s_{T_n}^{u_k} | s_{T_n}^{u_k}, a_{T_n}^{u_k}, \theta_{T_n-1})).
\]

In the above equation, we maximise the likelihood of the state information given the state representation and predicted teammate action information contained in each particle.

**Agent Modelling Loss Function.** While other methods use separate models for agent modelling and latent variable inference, the target action distribution estimation model, \( p_{\alpha}(a_{T_n}^{u_k} | e_{T_n}^{u_k}, s_{T_n}^{u_k}, \theta_{T_n}^{u_k}) \), is reused for agent modelling when using particle-based belief models. This reuse is motivated by how GPL’s agent modelling process introduced in Section 4.4 aims to estimate the target action distribution in the first place. Therefore, the negative log-likelihood loss is computed by assuming \( P_{\text{ag}} = \{\alpha\} \). The negative log-likelihood loss is then defined as:

\[
L_{\text{NLL}}^{P_{\text{ag}}}(D) = - \sum_{H_n \in D} \sum_{u_k \in U_n} \log\left( q_{P_{\text{ag}}}(a_{T_n}^{V_n}, u_k, \theta_{T_n}^{u_k}) \right),
\]

where the joint action log-likelihood is only evaluated over observed teammates’ joint actions.

**Reinforcement Learning Loss Function.** The CG-based model used in action value computation defined in Section 5.4.4 is parameterised by \( \eta \) such that \( P_{\text{val}} = \{\eta\} \). Assuming \( V^\text{A} \) denotes the set of possible joint actions of unobserved agents, the CG-based joint action value model is then trained to estimate the optimal joint action value function by optimising:

\[
L_{\text{RL}}^{P_{\text{val}}}(D) = \sum_{H_n \in D} \left( \sum_{u_k \in U_n} \frac{\exp(w_{T_n}^{u_k})}{2 \sum_{u_j \in U_n} \exp(w_{T_n}^{u_j})} \left( y_{P_{\text{val}}}(u_k, a_{T_n}^{V_n}) - y(u_k') \right)^2 \right),
\]
where,

\[
y_{\text{val}}(u_k, a^V_k) = \sum_{a^V \in A^V} Q_{\text{val}}(e^k_\alpha, s^k_\alpha, \theta^k_\alpha, a^V, a^{-V}) p_\alpha(a^{-V} | e^k_\alpha, s^k_\alpha, \theta^k_\alpha, T_n), \quad (5.49)
\]

is the estimated joint action value of agents visible to the learner at \( T_n \) based on the contents of particle \( u_k \). Specifically, \( Q_{\text{val}}(e^k_\alpha, s^k_\alpha, \theta^k_\alpha, a^V, a^{-V}) \) is computed via Equation 4.5 while \( p_\alpha(a^{-V} | e^k_\alpha, s^k_\alpha, \theta^k_\alpha, T_n) \) is evaluated based on Equation 4.8.

The target value for particle \( u_k \) is then defined based on \( u'_k \), which is the particle resulting from updating \( u_k \) based on \( o^n_{T_n+1} \) and \( a^{in}_{T_n} \) according to Section 5.4.2 excluding the particle sampling step. The target value is then defined as:

\[
y(u'_k) = r^p_{T_n} + \gamma \max_{\alpha'} \tilde{Q}(s^u_{T_n}, \theta^u_{T_n}, a'), \quad (5.50)
\]

with \( \tilde{Q}(s^u_{T_n}, \theta^u_{T_n}, a') \) evaluated according to Equation 4.11. Unlike in the RL loss under full observability, we consider the particle weights in the loss computation to allow less likely particles to have higher temporal difference errors.

### 5.6.2 Variational Autoencoder-based Belief Inference Models

**Belief Inference Loss Function.** The ELBO loss function that we define to train the variational autoencoder-based belief model is defined below:

\[
L_{\text{ELBO}}^\text{ELBO}(D) = - \sum_{H_n \sim D} \mathbb{E}_{z_{T_n} \sim q_{\text{inf}}(z_{T_n} | H_n)} \left[ \log(p_{\text{inf}}(B_{\text{obs}}(o^n_{T_n} | z_{T_n})) \right) \right.
\]

\[
+ \log \left( p_{\text{inf}}(a^V_{T_n} | z_{T_n}) \right] \right]
\]

\[
\left. - D_{KL}(q_{\text{inf}}(z_{T_n} | H_n) || p(z_{T_n})). \quad (5.53) \right]
\]

The distributions involved in the computation of this loss function are defined following the network architectures described in Section 5.3.2, which are parameterised by \( P_{\text{inf}} = \{ \alpha, \beta, \gamma \} \). To enable backpropagation through the sampling operation on \( q(z_{T_n} | H_n) \), we use reparameterisation tricks that are commonly used in optimising variational autoencoders (Kingma and Welling [2013]).

**State Reconstruction Loss Function.** Like in the particle-based inference method, we define another model parameterised by \( P_{\text{sr}} = \{ \zeta \} \) to parameterise the state reconstruction distribution, \( p_{\text{sr}}(B_{\text{obs}}(s^p_{T_n}) | z_{T_n}) \). Given representations sampled from the encoder, \( z_{T_n} \), the state reconstruction loss function is defined as:

\[
L_{\text{ELBO} \cup P_{\text{sr}}}(D) = - \sum_{H_n \sim D} \mathbb{E}_{z_{T_n} \sim q_{\text{inf}}(z_{T_n} | H_n)} \left[ \log(p_{\text{sr}}(B_{\text{obs}}(s^p_{T_n}) | z_{T_n})) \right]. \quad (5.54)
\]
Agent Modelling Loss Function. Agent modelling under the VAE-based model is done via the action prediction component of the decoder, which in our description at Section 5.5.2 is parameterised by $\gamma$. This model is chosen for agent modelling since it also aims to predict teammates’ joint actions. Assuming $P_{ag} = \{\gamma\}$, the loss function of this model is defined as:

$$L_{NLL}^{P_{ag}}(D) = - \sum_{H_n \in D} \mathbb{E}_{z_T^n \sim q(z_T^n|H_n)} \left[ p_{P_{ag}}(a^V_{T_n}|z_T^n) \right]$$

(5.55)

Reinforcement Learning Loss Function. As in GPL, we define a CG-based model to estimate the joint action values of the learner. Assuming that the parameters of this model are denoted as $\delta$, this model must be trained to estimate the joint action value given the variational parametric distribution, $q(z_t|H_t)$. Since exactly computing $q(z_t|H_t)$ is generally intractable, we use a Monte Carlo approach for training this model. Under this approach, we sample $m$ vectors from $q(z_t|H_t)$ such that:

$$z_1^t, z_2^t, ..., z_m^t \overset{iid}{\sim} q(z_t|H_t).$$

(5.56)

The sampled $z_t$ are subsequently used as input to the CG model, which loss function for joint action value modelling is subsequently computed as:

$$L_{RL}^{P_{val}}(D) = \sum_{H_n \in D} \left( \sum_{k=1}^m \left( \frac{p(z_k^{T_n}|H_n)}{2\sum_{l=1}^m p(z_l^{T_n}|H_n)} \left( y_{P_{val}}(z_k^{T_n}, a^V_{T_n}, a^{-V}_{T_n}) - y(z_k^{T_n}) \right)^2 \right) \right)$$

(5.57)

assuming $P_{val} = \{\delta\}$. In Equation 5.57, the predicted joint action value of observed teammates’ joint actions is defined as:

$$y_{P_{val}}(u_k, a^V_{T_n}, a^{-V}_{T_n}) = \sum_{a^{-V} \in A^{-V}} Q_{P_{val}}(z_k^{T_n}, a^V_{T_n}, a^{-V}) p_{a^{-V}}(a^{-V}|z_k^{T_n}),$$

(5.58)

which is similar to the predicted value under particle-based approaches. Finally, the target value is defined as:

$$y(u'_k) = r_k^n + \gamma \max_{a'} \bar{Q}(Z_k^{T_n}, a'),$$

(5.59)

with $\bar{Q}(Z_k^{T_n}, a')$ computed according to Equation 5.42.

5.6.3 Representation-based Models

Belief Inference Loss Function. The encoder and decoder are trained to minimise the following reconstruction loss function:

$$L^{RECONS}_{P_{inf}}(D) = - \sum_{H_n \in D} \left( \| B_{P_{inf}}^{\rho}(\rho_{T_n}) - B_{obs}(\sigma_{T_n}^p) \|^2 + \log(p_{P_{inf}}(a^V_{T_n}|\rho_{T_n})) \right)$$

(5.60)
assuming that $P_{inf} = \{\alpha, \beta\}$ are the parameters of the encoder and decoder model introduced in Section 5.3.1. The first term in Equation 5.60 ensures the encoder produces representations, $\rho_{T_n}$, containing observed teammate information. The second term enforces $\rho_T$ to contain information about teammates’ actions. As in the ELBO loss for variational autoencoders, the above loss function enables the encoder to produce representations that contain information on teammates’ behaviour during interaction.

**State Reconstruction Loss Function.** Similar to the optimisation of our VAE-based model, we define a state reconstruction model parameterised by $P_{st} = \{\zeta\}$. This model is used to reconstruct the state from the representation produced by the encoder. Both the autoencoder and the state reconstruction model are then trained to minimise the following loss function:

$$L_{SR}^{P_{inf} \cup P_{st}}(D) = -\sum_{H_n \in D} ||P_{pred}^{P_{inf} \cup P_{st}}(\rho_{T_n}) - B_{obs}(s_{T_n}^n)||^2.$$  \hspace{1cm} (5.61)

**Agent Modelling Loss Function.** Given $\rho_{T_n}$ produced by the encoder, GPL’s agent and joint action model are trained to estimate the learner’s action value function. We use the decoder’s action prediction component for agent modelling since it is also designed to predict teammates’ actions. Assuming $P_{ag} = \{\beta\}$, the agent model is subsequently trained to predict observed teammates’ actions by minimising the following loss function:

$$L_{NLL}^{P_{ag}}(D) = -\sum_{H_n \in D} \log(p_{P_{ag}}(a_{V_{T_n}}|\rho_{T_n})).$$  \hspace{1cm} (5.62)

**Reinforcement Learning Loss Function.** We train the joint action value model by optimising the temporal difference error defined below:

$$L_{RL}^{P_{val}}(D) = \sum_{H_n \in D} \left(y_{P_{val}}(\rho_{T_n}^k, a_{V_{T_n}}^n) - y(\rho_{T_n}^k)\right)^2.$$  \hspace{1cm} (5.63)

Given the parameters of the joint action value model $P_{val} = \{\delta\}$, the predicted and target joint action value are evaluated following Equation 5.58 and 5.59.

### 5.7 GPL Pseudocode Under Partial Observability

This section focuses on providing pseudocodes that illustrate how the learner updates its belief representations and uses belief representations to estimate the learner’s optimal action-value function under partial observability. Note the general training and decision-making procedure under partial observability highly resembles their respective counterparts under full observability provided in Algorithm 4. Therefore, we avoid
rewriting the entire training and decision-making pseudocode by highlighting the main
differences between instances of the GPL algorithm under the partial and fully observ-
able scenarios.

The GPL algorithm in partial and fully observable environments has three main
differences. The first two differences relate to how action values are computed follow-
ing Algorithm 1. The final difference relates to the additional loss functions to train
the belief inference models under partially observable scenarios.

The first difference between the action value computation in fully and partially ob-
servable scenarios relates to how inputs for the joint action value and agent model are
computed. Under full observability, the agent and joint action value model input repre-
sentations are computed via an LSTM. By contrast, input representations for the joint
action value and agent model are computed via the belief inference model introduced
in Section 5.1. This process of computing the input representation given our belief
inference models is provided in Algorithm 5.

The second difference between action value computation under partial and fully
observable scenarios is how the inferred representations are computed for action value
computation. In the GPL algorithm under full observability, this process is illustrated
by the lines following the calls to the LSTM in the third line of Algorithms 2 and 3.
However, the action value computation under partial observability depends on the be-
belief inference model being used. We illustrate how different belief inference models
use their outputted representations for decision-making in Algorithm 7. Note that re-
gardless of the belief inference method, the way an action-value function is computed
for a single sampled representation is the same, indicated by Algorithm 6.

Finally, the loss function is the last difference between the pseudocode for training
and decision-making under full and partially observable scenarios. Under full observ-
ability, we do not have loss functions associated with belief inference. However, we
now incorporate this loss function for training the belief inference model according to
the losses defined in Section 5.6

5.8 Partially Observable Open Ad Hoc Teamwork Ex-
periments

This section describes experiments performed using the belief inference methods in-
troduced in this chapter. We evaluate the methods in several open ad hoc teamwork
Algorithm 5 Belief Inference

Input:
Learner’s observation, $o_t$, and previous action, $a^t_{t-1}$
The belief inference algorithm, $\text{alg} \in \{\text{PF, AE, VAE}\}$,
Representations resulting from the previous step,

$$
\rho_{t-1} = \begin{cases}
\{(a^u_{t-1}, s^u_{t-1}, \theta^u_{t-1}, w^u_{t-1}) | u_k \in U_{t-1}\}, & \text{if } \text{alg} = \text{PF} \\
(c_{t-1}, h_{t-1}), & \text{if otherwise}
\end{cases}
$$

1: function BELIEF\_INFERENCE($o_t, a^t_{t-1}, \text{alg}, \rho_{t-1}$)
2: \hspace{1em} if $\text{alg} = \text{PF}$ then
3: \hspace{2em} Sample $K$ particles, $\tilde{U}_{t-1} = \{u_1, u_2, ..., u_K\}$, from $\rho_{t-1}$ with,
4: \hspace{2em} $u_1, u_2, ..., u_K \overset{\text{i.i.d.}}{\sim} \text{Categorical}(\text{Softmax}(w^u_{t-1} | u_k \in \rho_{t-1}))$
5: \hspace{2em} for $u_k \in U_{t-1}$ do
6: \hspace{3em} $a^u_{t-1} \sim q_\alpha(a^u_{t-1} | s^u_{t-1}, \theta^u_{t-1}, a^i_{t-1}, o_t)$ \Comment{Action Inference}
7: \hspace{3em} $s^u_{t} \sim p_\beta(s^u_{t} | s^u_{t-1}, a^u_{t-1}, a^i_{t-1}, o_t)$ \Comment{State Inference}
8: \hspace{3em} $\theta^u_t = f_\delta(s^u_t, \theta^u_{t-1}, a^u_t, a^i_{t-1}, o_t)$ \Comment{Type Update}
9: \hspace{3em} Compute $w^u_{t-1,\alpha}$ and $w^u_{t-1,\beta}$ following Equations 5.13 and 5.14
10: \hspace{3em} $w^u_{t} = \log(p_\zeta(o_t | s^u_t, a^u_{t-1})) + w^u_{t-1,\beta} + w^u_{t-1,\alpha}$ \Comment{Particle Weight Update}
11: \hspace{2em} end for
12: \hspace{1em} $U_t = \{(a^u_{t-1}, s^u_t, \theta^u_t, w^u_t) | u_k \in \tilde{U}_{t-1}\}$
13: \hspace{1em} return: $U_t, U_t$
14: else
15: \hspace{2em} if $\text{alg} = \text{VAE}$ then
16: \hspace{3em} $\mu_t, \Sigma_t, (c_t, h_t) = \text{Encoder}_\alpha(a^i_{t-1}, o_t, \rho_{t-1})$ \Comment{Based on Section 5.5.2}
17: \hspace{3em} $z_1, z_2, ..., z_K \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu_t, \Sigma_t)$ \Comment{Sample $K$ representations}
18: \hspace{3em} $Z_t = \{(z_1, p(z_1 | \mu_t, \Sigma_t)), (z_2, p(z_2 | \mu_t, \Sigma_t)), ..., (z_K, p(z_K | \mu_t, \Sigma_t))\}$
19: \hspace{3em} return: $Z_t, (c_t, h_t)$
20: else
21: \hspace{3em} $z_t, (c_t, h_t) = \text{Encoder}_\alpha(a^i_{t-1}, o_t, \rho_{t-1})$ \Comment{Based on Section 5.3.1}
22: \hspace{3em} return: $z_t, (c_t, h_t)$
23: end if
24: end function
Algorithm 6 Single Sample Action Value Computation Under Partial Observability

Input:
- Input representation, \( \rho \).
- Joint-action value model parameters, \( P_{val} \).
- Agent model parameters, \( P_{ag} \).

1: function \( QV_{\text{PART}}(\rho, P_{val}, P_{ag}) \)
2: \( \forall j, \bar{n}_j \leftarrow (RFM_{P_{ag}}(\rho))^j \)
3: \( \forall j, q_{P_{ag}}(.|\rho) \leftarrow \text{Softmax}(MLP_{P_{ag}}(\bar{n}_j)) \)
4: \( \forall j, a^j, Q_{P_{val}}^{j}(a^j|\rho) \leftarrow \text{MLP}_{P_{val}}(p^j, \rho^j)(a^j) \)
5: \( \forall j, a^j, a^k, Q_{P_{val}}^{j,k}(a^j, a^k|\rho) \leftarrow \text{MLP}_{\delta}(p^j, p^k, \rho^j)(a^j, a^k) \)
6: Compute \( \bar{Q}(\rho, a^j) \) using Equation 4.10
7: \( \bar{Q}(\rho, .) \leftarrow \text{MARGINALIZE}(q_{P_{ag}}(.|\rho), Q_{P_{val}}^{\text{local}}(.|\rho), Q_{P_{val}}^{\text{global}}(.|\rho)) \)
8: return \( \bar{Q}(\rho, .) \)
9: end function

We first describe the environments and algorithms used in our evaluation (Section 5.8.1). We then present a performance of our algorithms followed by a reconstruction analysis that seeks to evaluate the performance of the different belief methods.

5.8.1 Experimental Setup

We describe the environments and the algorithms used for our evaluation in Sections 5.8.1.1 and 5.8.1.2. The open process underlying the environments used in this chapter’s experiments follows the open process used in the fully observable setup. Similarly, the types of teammates used in this experiment follow that of the fully observable setup.

5.8.1.1 Environments

We utilised two environments previously used in the fully observable experiments, for which we introduce partial observability using different observation functions. Finally, we also incorporate a new environment for the partially observable case called Penalised Cooperative Navigation (PCN).

Level-Based Foraging. In LBF we induce partial observability by only allow-
Algorithm 7 Action Value Computation Under Partial Observability

Input:
The belief inference algorithm, alg ∈ {PF, AE, VAE}.
Joint-action value model parameters, $P_{val}$.
Agent model parameters, $P_{ag}$.

Representations resulting from the belief inference model,
\[
\rho_t = \begin{cases} 
\{(a_{t-1}^u, s_t^u, \theta_t^u, w_t^u) | u \in U_t\}, & \text{if alg = PF} \\
\{(z_1, p(z_1 | \mu_t, \Sigma_t)), \ldots, (z_K, p(z_K | \mu_t, \Sigma_t))\}, & \text{if alg = VAE} \\
z_t, & \text{if otherwise}
\end{cases}
\]

1: function QV.P.OBS(alg, $\rho_t$, $P_{val}$, $P_{ag}$)
2: if alg = AE then
3: return QV.PART($\rho_t$, $P_{val}$, $P_{ag}$)
else
4: if alg = PF then
5: for $u_k \in \rho_t$ do
6: $x^{u_k} \leftarrow \text{CONCATENATE}(e^{u_k}_t, s^{u_k}_t, \theta^{u_k}_t)$
7: $\bar{Q}(x^{u_k}_t, s^{u_k}_t, \theta^{u_k}_t, \ldots) \leftarrow \text{QV.PART}(x^{u_k}_t, P_{val}, P_{ag})$
8: end for
9: return $\sum_{u_k \in U_t} \left( \frac{\exp(w^{u_k}_t)}{\sum_{u_j \in U_t} \exp(w^{u_j}_t)} \right) \bar{Q}_{\rho_t}(x^{u_k}_t, s^{u_k}_t, \theta^{u_k}_t, \ldots)$
else
10: for $(z_k, p(z_k | \rho_t, \Sigma_t)) \in \rho_t$ do
11: $\bar{Q}(z_k, \ldots) \leftarrow \text{QV.PART}(z_k, P_{val}, P_{ag})$
12: end for
13: return $\sum_{(z_k, p(z_k | \rho_t, \Sigma_t)) \in \rho_t} \frac{\bar{Q}(z_k^k, \ldots) p(z_k | \rho_t, \Sigma_t)}{\sum_{(z_k, p(z_k | \rho_t, \Sigma_t)) \in \rho_t} p(z_k | \rho_t, \Sigma_t)}$
end if
14: end function

In the learner to see objects and teammates within a specific region surrounding the learner’s current grid. For this particular test setup, we utilised a grid world of size $12 \times 12$ and only allow the learner to observe entities within a $5 \times 5$ grid centred in the learner.

**Wolfpack.** In Wolfpack, partial observability is induced by restricting the learner
to only observe agents and prey whose Manhattan distance is less than a certain value relative to itself. We set the grid world as a $10 \times 10$ square and limit the learner’s observation to entities within a Manhattan distance of 3 from itself.

**Penalised Cooperative Navigation (PCN).** Similar to the cooperative navigation environment (Tacchetti et al., 2019), multiple players must navigate through a $12 \times 12$ grid world to simultaneously cover two destination grids to get a reward of 1. However, the learner is given a $-0.2$ penalty if it arrives at a destination without other teammates covering the other. We make reasoning through partial observability a necessity by frequently positioning the destination grids far away from each other. To avoid penalties, the player must reason whether teammates are about to arrive at a destination outside its observation. After a pair of agents arrive at the destinations, we randomly choose a new pair of destination grids. Like LBF, the learner can only see the destination grids or teammates if they are inside a $5 \times 5$ region surrounding the learner.

### 5.8.1.2 Evaluated Algorithms

We present the different algorithms developed based on the belief representation methods described in this chapter. Each of these algorithms uses GNNs to produce representations that characterise the latent environment state, which is inputted to the joint action value and agent model for optimal action-value function estimation. Table 5.1 summarises the different algorithms by describing their main components.

**Representation-based State Inference (AE-GPL).** This algorithm utilise an autoencoder architecture, as presented in Section 5.3, to create the AE-GPL algorithm. AE-GPL learns an embedding $z_t$ that contains information about the teammates’ policies and the environment dynamics. This embedding $z_t$ can then be used for decision-making by the learner. We named this algorithm Autoencoder GPL (AE-GPL).

**Particle-based Belief (PF-GPL).** We introduce the particle filter graph-based belief and policy learning (PF-GPL) algorithm, which utilises the particle-based representation as presented in Section 5.4. PF-GPL allocates separate representations to model the different latent variables required for decision-making. We formulate different ablations to identify the differences that result from utilising different numbers of particles in the belief inference of SMC-based methods. We run PF-GPL ablations with ten (PF-GPL-10), five (PF-GPL-5) and one particle (PF-GPL-1) to see the effects of using less and even a single sampled vector to represent the agent’s belief.

**Variational Autoencoder-based Belief (VAE-GPL).** Variational Autoencoder GPL (VAE-GPL) is an algorithm based on the method presented in Section 5.5.
Table 5.1: Evaluated belief inference algorithms. Categorisation of the belief inference algorithms evaluated in this work based on the usage of separate representations for different latent variables, the addition of state reconstruction loss for training, and approximate belief inference method being used.

<table>
<thead>
<tr>
<th>Models</th>
<th>Separate variable representation</th>
<th>State reconstruction</th>
<th>State Inference method</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPL-Q</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AE-GPL</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PF-GPL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>VAE-GPL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

utilises a variational autoencoder to maintain a distribution of latent variables $z_t$ that encodes the belief about the environment’s current state. We also train VAE-GPL to reconstruct the state information, which we assume to be known during training.

Graph-based Policy Learning (GPL-Q). While it assumes full state observability, GPL-Q can still be used under partial observability. We apply GPL-Q in our experiments by treating the learner’s observations as input states from the environment. Following the effectiveness of RNNs for learning policies for POMDPs (Hausknecht and Stone 2015), GPL-Q’s RNN-based type inference network should still facilitate the learning of reasonable policies from $o_{\leq t}$ and $a^{d}_{<t}$. Unlike other evaluated algorithms, GPL-Q is not trained to reconstruct the state information during training.

5.8.2 Partially Observable Open Ad Hoc Teamwork Results

The returns obtained by the proposed methods in the partially observable open ad hoc teamwork experiments are provided in Figure 5.5. We see that the autoencoder and variational autoencoder-based methods in all three environments learn to achieve significantly higher returns than other methods. This is particularly true in the cooperative navigation environment, where PF-GPL-based methods achieve a return close to zero. Nonetheless, PF-GPL-based methods improve their returns in LBF and Wolfpack as the number of particles used during inference increases. This aligns with previous results from other particle-based methods (Albrecht and Ramamoorthy 2016), which demonstrates the need for using a larger number of particles to increase belief representation accuracy.

The suboptimal performance of PF-GPL in Figure 5.5 suggests that the proposed particle-based belief representation cannot generate useful representations for decision-making. Unlike the other methods evaluated in our experiment, PF-GPL is the only
method that updates the different estimates for each latent variable without using recurrent neural networks. Particularly, the particle-based belief inference method assumes that the contents of a particle at time $t$ have sufficient information regarding the learner’s previous experience, $H_t$. These experiments show that using a recurrent neural network to produce a single representation of all latent variables may be the key to achieving higher performance in partially observable open ad hoc teamwork problems.

AE-GPL and VAE-GPL are the only methods consistently achieving high returns in all three environments. In the Cooperative Navigation environment where reasoning capabilities on unobserved teammates are most important, AE-GPL and VAE-GPL still achieve high returns. AE-GPL and VAE-GPL’s significantly higher returns suggest the importance of using recurrent neural networks for latent variable inference and observation reconstruction to create useful representations for decision-making. Other methods not equipped with observation reconstruction, such as GPL-Q cannot
5.8. Partially Observable Open Ad Hoc Teamwork Experiments

Table 5.2: Partially observable open ad hoc teamwork results (testing). We show the average and 95% confidence bounds during testing utilising 8 seeds. The data was gathered by averaging the returns at the checkpoint which achieved the highest average performance during training. We highlight in bold the algorithm with the highest average returns.

<table>
<thead>
<tr>
<th>Env.</th>
<th>GPL-Q</th>
<th>AE-GPL</th>
<th>PF-GPL-10</th>
<th>PF-GPL-5</th>
<th>PF-GPL-1</th>
<th>VAE-GPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBF</td>
<td>1.03±1.70</td>
<td>0.95±1.40</td>
<td>0.82±1.42</td>
<td>0.76±0.93</td>
<td>0.61±1.16</td>
<td><strong>1.13±1.80</strong></td>
</tr>
<tr>
<td>Coop</td>
<td>0.22±1.26</td>
<td><strong>1.02±1.47</strong></td>
<td>0.05±0.27</td>
<td>0.06±0.38</td>
<td>0.05±0.35</td>
<td>0.71±1.41</td>
</tr>
</tbody>
</table>

consistently achieve high returns.

We also see that VAE-GPL is the best-performing method in LBF, with GPL-Q and AE-GPL achieving comparable returns. This tendency is maintained in Wolfpack as can be seen in Figure 5.5. It is important to note that GPL-Q achieves comparable performance despite not having models specifically designed for belief inference. Although this may seem surprising, note that the RNN-based type inference model of GPL-Q still enables the discovery of important information for decision-making based on the sequence of observations experienced by the learner. We can view the changing number of teammates resulting from the learner’s partial observability as another open process, which the learner can still solve as long as the sequence of observations contains useful information for action selection.

In contrast to its results in LBF and Wolfpack, GPL-Q performs poorly in cooperative navigation. This is because the sequence of observations perceived by the learner contains the least helpful information compared to other environments. It is important to note that the most critical information in cooperative navigation is whether another teammate is positioned nearby another destination grid, which is usually unobserved. As such, methods with an additional state reconstruction loss will undoubtedly produce better decision-making representations than GPL-Q.

Similarly to our generalisation experiments under the fully observable setting, we present the generalisation capabilities of the agents to different numbers of teammates in Table 5.2. The results show that none of the methods outperforms each other during generalisation even when they achieve similar returns during training. Unsurprisingly, methods that achieve low returns during training, such as PF-GPL, will also achieve subpar performance when generalising to different open processes.

The high return variance in our generalisation experiments demonstrates a potential issue with our proposed belief inference models. Training the belief inference
Figure 5.6: **Action reconstruction accuracy.** We evaluate the log probability between the predicted actions and the true actions taken by the teammates. We ran the inference modules for each algorithm over a fixed episode in which actions were predetermined. We evaluated the log probability \( n \) times over each checkpoint’s fixed episode and reported the mean and 95% confidence bounds.

5.8.3 Reconstruction Results

In this section, we evaluate the reconstruction capabilities of the methods proposed in Section [5.8.1.2](#). We do this evaluation for two reasons. First, we want to examine...
Figure 5.7: **State reconstruction accuracy.** We evaluate the log probability between the predicted state and the true state of the system. We ran the inference modules for each algorithm over a fixed episode in which actions were predetermined. We evaluated the log probability $n$ times over each checkpoint’s fixed episode and reported the mean and 95% confidence bounds.

whether the methods can represent useful information for decision-making. Second, we also aim to elucidate which learned information is most useful in improving the returns of the learner. This evaluation is done on the environments defined in Section 5.8.1.1.

The reconstruction evaluation was done over a single episode. We collect an episode of interaction data $H = \{o_t, a_t\}_{t=1}^T$, by executing the policy resulting from the algorithm with the highest training returns in each respective environment. At every training checkpoint, we utilise the single-episode interaction data to evaluate each method’s reconstruction capabilities for different measures such as the environment state, teammates’ joint actions, and teammates’ existence.

The resulting reconstruction performance for teammates’ joint actions, state reconstruction, and teammates’ existence reconstruction are provided in Figure 5.6, Figure 5.7, and Figure 5.8 respectively. To evaluate action reconstruction, at each checkpoint we report the average log-likelihood of all teammates’ joint actions, which includes teammates that are not observed by the learner. We then evaluate the state
reconstruction capabilities of the methods by reporting the log probability they assign to the unobserved state of the environment. Assuming existing teammates are denoted by a binary value of one while non-existent teammates are assigned a value of zero, we report the sum of the squared error between the predicted and real teammate existence for all agents.

Among the evaluated measures, the capability of the methods in teammate action prediction is the best indicator of their achieved returns during training. This is mainly because a method incapable of accurately predicting the teammates’ joint actions will lead the learner to produce worse action value estimates. Following its significantly worse action prediction performance compared to other methods, it is unsurprising to see PF-based methods’ failure in achieving high returns during training. Meanwhile, GPL-Q, AE-GPL, and VAE-GPL produce higher returns resulting from having better teammate joint action prediction.
An improved state reconstruction capability of a method also leads towards improved returns during training. While the state reconstruction performance of the methods under this measure is similar in LBF and FortAttack, AE-GPL is significantly better than other methods in cooperative navigation. Improving state prediction capabilities in cooperative navigation is crucial for producing high returns, since estimating whether teammates are close to an unobserved destination grid is the only way for the learner to avoid being penalised. As a result, AE-GPL outperforms other methods in cooperative navigation even if it performs similarly with GPL-Q and VAE-GPL in action reconstruction. Finally, the results suggest that reconstructing agent existence is the least important for producing high returns during training. PF-based methods significantly achieve the lowest squared error for this particular measure. Despite its ability to accurately predict agents’ existence, its inability to accurately predict the state and joint actions of teammates prevents PF-based methods from achieving higher returns.

5.9 Chapter Summary

This chapter detailed our work on addressing the problem of open ad hoc teamwork under partial observability. Our methods for solving this problem explored how different methodologies could provide a belief estimate of the state. We specifically evaluated three belief estimation methodologies: autoencoder-based, variational autoencoder-based, and particle-based belief inference methods. Similar to our experiments in the fully observable setup from the previous chapter, we evaluate our proposed algorithms in three different environments in which agents have only partial access to the system’s state during evaluation.

The results of our experiment show that variational autoencoder methods can outperform the other baselines in LBF and Wolfpack, while autoencoder-based methods surpass the performance of other belief inference methods for Cooperative Navigation. After further analysing the latent variables inferred by each method, we discover that methods capable of improved accuracy in predicting the teammate’s actions can achieve higher returns. An improved state estimation, which we observe in autoencoder-based methods, also explains the difference in performance for the Cooperative Navigation environment.
Chapter 6

Generating Teammates For Ad Hoc Teamwork

The previous chapters defined AHT methods requiring predefined teammate types to train a learner. When collaborating with teammates displaying significantly different behaviour than teammates encountered during training, a problem arises for GPL since it may produce type vectors that do not contain helpful information for decision-making. Such type vectors can mislead GPL’s agent and joint action value model into producing erroneous action likelihood and joint action value estimates. Therefore, ensuring the behavioural diversity of training teammates is crucial to improve the robustness of a learner when dealing with previously unseen teammates.

The importance of providing a diverse range of training teammate types is not limited to GPL. Regardless of their action selection method based on interactions with teammates during training, the behavioural diversity of training teammate types significantly affects the robustness of a learner produced by any type-based ad hoc teamwork (AHT) method. For instance, expert policy methods (Chen et al., 2020; Santos et al., 2021; Barrett et al., 2017) will not have an expert policy that can be consulted for action selection when collaborating with a teammate unless that teammate has been encountered during training.

Following the importance of diverse training types for AHT training in type-based methods, this chapter provides the details of Best-Response Diversity (BRDiv), which is our proposed method to generate a set of diverse teammate policies for AHT training. Unlike previous approaches for teammate generation in AHT or other related problems (Lupu et al., 2021; Xing et al., 2021; Lucas and Allen, 2022), BRDiv prevents the emergence of teammates with superficial differences by ensuring that each
generated policy requires a different best-response policy for effective collaboration. We cover the notion of the superficial difference between generated policies and its potential detriment for training a robust AHT learner in Section 6.1. We then start our description of BRDiv in Section 6.2 by outlining an optimised diversity metric that facilitates the discovery of $\Pi^{\text{train}} = \{\pi^1, \pi^2, \ldots, \pi^K\}$, which is a collection of AHT teammate policies that require distinct best-response policies. Sections 6.3 and 6.4 then conclude our description of BRDiv by introducing a MARL-based technique to optimise our proposed diversity metric alongside its pseudocode.

Our experiments’ results indicate BRDiv’s effectiveness in improving the robustness of an AHT learner. By training against teammates requiring different best-response policies for effective collaboration, we show that BRDiv encourages a learner to acquire a more comprehensive range of collaboration strategies that improve the robustness of the learner. Meanwhile, a learner trained using teammates generated by other evaluated methods only manages to cover a subset of the collaboration strategies learned by BRDiv-based learners.

### 6.1 Desirable Diversity For Ad Hoc Teamwork Training

Before we discuss the details of BRDiv, we first discuss a suitable notion of diversity when generating AHT teammate types. Our notion of diversity is motivated by the deficiencies of previous automated teammate generation methods for AHT or other similar problems such as zero-shot coordination (ZSC) (Hu et al., 2020), which we previously discussed in Section 2.5.1. Existing teammate generation approaches for AHT and ZSC (Lupu et al., 2021; Xing et al., 2021; Lucas and Allen, 2022) rely on diversity metrics that encourage different teammate types to produce distinct trajectories when collaborating with other agents. Metrics optimised by these approaches are typically based on the maximisation of information-theoretic divergence, which encourages each generated type’s trajectory or action distribution to be as different as possible.

The problem with teammate generation methods based on information-theoretic diversity maximisation lies in how it may produce teammates with superficial differences, which does not encourage the emergence of a robust AHT learner that can effectively collaborate with previously unseen teammates as formally defined in Section 3.3. To illustrate behaviour with superficial differences, consider the problem of creating training teammate types for a soccer game. Under this soccer setup, superficial
Figure 6.1: **Potential teammate types generated from information-theoretic diversity maximisation.** In all soccer pitch visualisations except the one on the bottom right, we provide three example teammate behaviours with high information-theoretic diversity in a 3 vs 3 soccer environment. Each of these three visualisations highlights the behaviour of a teammate that favours dribbling and shooting, with the favoured dribbling trajectory of each teammate indicated by the green dashed arrow. The soccer pitch visualisation on the bottom right then shows the behaviour acquired by a learner that encounters the previously mentioned teammates during training. Despite the diversity in these teammates’ trajectories, an AHT learner will only acquire the skill to pass the ball to teammates, which is indicated by the blue solid line, since it is sufficient for effective collaboration with all three teammates.

Differences can be produced by creating teammates displaying different trajectories to execute the same style of play. Figure 6.1 exemplifies this as three teammate types that like to dribble with different trajectories in an information-theoretic sense.

Superficially different behaviours provide a challenge to AHT training since they do not encourage improved learner robustness. In the example provided in Figure 6.1, effective collaboration with all three teammate types can be achieved by a common strategy of passing the ball to these teammates. A learner trained to collaborate with these teammates will only acquire expertise in passing the ball to teammates while neglecting other essential skills in soccer. This prevents the emergence of effective
Figure 6.2: **A more useful set of teammate types for AHT training.** We provide an example of a more useful set of teammate policies to improve the robustness of a learner at 3 vs 3 soccer in all three pitch visualisations except the bottom right one. Compared to Figure 6.1, this visualisation introduces two novel teammate types. The top right visualisation introduces a teammate behaviour that favours passing into space, in which the preferred passing trajectory is highlighted by the solid blue arrow. Effective collaboration against this newly introduced behaviour requires a policy favouring off-the-ball movement and finishing. The bottom left visualisation also introduces a different behaviour favouring passing the ball to a teammate and moving backwards to defend, with teammates’ off-the-ball movement trajectory highlighted by the black solid arrow. Against this type of teammate, a policy favouring dribbling the ball to carry it closer to the goal and then shooting is required for effective collaboration. Finally, the bottom right pitch visualisation illustrates the skills acquired by a learner from training against this ideal set of teammates. Given enough interactions with these teammates, the learner should acquire passing, off-the-ball movement, and dribbling skills.

collaboration when the learner has to interact with teammates where passing skills alone do not help, such as when interacting with another teammate that likes to pass. Note that despite this ineffectiveness in improving a learner’s robustness, prior teammate generation methods will potentially generate these teammate types since their behaviours still display diverse trajectories.
In the context of our soccer example, a more useful set of training teammate types is exemplified by Figure 6.2. The training teammate types depicted in Figure 6.2 are experts in different skills in soccer, such as dribbling, defence, and passing. Compared to the set of generated teammates illustrated in Figure 6.1, different strategies are required to effectively collaborate with each generated teammate type in this example. For instance, dealing with a defensive-minded teammate requires the learner to pick up a strategy geared towards dribbling and scoring. Meanwhile, collaboration with a creative teammate that loves to pass and create opportunities requires the learner to focus on off-the-ball movement and finishing skills.

Regardless of the AHT method used for training, using a set of training teammates requiring different collaborative strategies facilitates the design of a more robust learner. For AHT methods based on deep reinforcement learning (Rahman et al., 2021; Papoudakis et al., 2021a; Zintgraf et al., 2021), the policy models trained in these methods will associate each teammate’s type vector to a different collaborative skill. Meanwhile, this also facilitates expert policy AHT methods’ (Chen et al., 2020; Santos et al., 2021; Barrett et al., 2017) acquisition of expert policies corresponding to a wider range of useful behaviours for collaboration.

Learning a broader range of skills then improves the robustness of a learner by endowing the learner with a more comprehensive library of behaviours to effectively collaborate with any teammate type. In the previously mentioned soccer example, Figure 6.2 illustrates how a learner acquires skills such as dribbling, passing, and off-the-ball movement by training against teammates requiring different collaborative strategies. In this work, we use the significance of having different strategies for effective collaboration as a motivation to formulate the diversity metric optimised by BRDiv, which we outline in Section 6.2.

### 6.2 Best-Response Diversity Metric

This section defines a diversity metric that we utilise to develop teammate policies requiring different collaborative strategies for effective collaboration. Our description of BRDiv assumes that only two agents exist in the environment. Nevertheless, extending our proposed diversity metric to environments with more than two agents is straightforward.

BRDiv aims to generate a set of diverse policies for AHT training, \( \Pi^{\text{train}} = \{ \pi^1, \pi^2, \ldots, \pi^K \} \), where similar best-response policies cannot be used to effectively collaborate...
Chapter 6. Generating Teammates For Ad Hoc Teamwork

with different generated teammate types from $\Pi^{train}$. Therefore, defining a metric that quantifies the effectiveness of two agents’ policies when collaborating with each other is a crucial first step in formulating our diversity metric. We measure the effectiveness of two policies when collaborating via their expected returns, which is inspired by our notion of robust collaboration introduced in Section 3.3.2. Assuming that agent $j$ and $k$ are interacting with each other based on policies, $\pi^j(a_j^1|H_j^t)$ and $\pi^k(a_k^1|H_k^t)$, that are conditioned on their respective observation-action history $H_j^t$ and $H_k^t$, this return-based effectiveness measure is defined as:

$$V_{j,k}(H_j^t, H_k^t) = \mathbb{E}_{a_1^T \sim \pi^j, a_2^T \sim \pi^k} \left[ \sum_{T=t}^{\infty} \gamma^{T-t} R(s_T, \langle a_1^T, a_2^T \rangle) \bigg| H_j^t, H_k^t \right].$$  (6.1)

This return-based effectiveness measure provides a foundation for defining an optimised diversity metric to achieve the goal of BRDiv. Denoting the best-response to $\pi^k$ by $\pi^{-k,*}$, and the set of best response policies to each policy in $\Pi^{train}$ by $\text{BR}(\Pi^{train})$, we use Equation 6.1 to evaluate the effectiveness of $\pi^-k,* \in \text{BR}(\Pi^{train})$ when collaborating with $\pi^j \in \Pi^{train}$. Given a pair of observation-action histories, $H_j^1$ and $H_k^1$, we arrange the measured cooperative effectiveness between all possible agent-best response policy pairs, $(\pi^j, \pi^-k,*) \in \Pi^{train} \times \text{BR}(\Pi^{train})$, into a $K \times K$ cross-play matrix, $C_{\Pi^{train}, \text{BR}(\Pi^{train})}(H_j^1, H_k^1)$, whose elements are defined as:

$$C_{\Pi^{train}, \text{BR}(\Pi^{train})}(H_j^1, H_k^1) = V_{j,-k,*}(H_j^1, H_k^1).$$  (6.2)

Our proposed diversity metric is based on the intuition that a good $\Pi^{train}$ to ensure the learner’s robustness must possess two characteristics. First, the cross-play matrix of $\Pi^{train}$ must have high values on its diagonal elements to ensure that each $\pi^j \in \Pi^{train}$ interacts effectively with its associated best-response policy, $\pi^-j \in \text{BR}(\Pi^{train})$. This characteristic also prevents the emergence of teammate policies producing low returns, which no reward-optimising agent would have a reason to use in an environment. Second, the off-diagonal elements of $C_{\Pi^{train}, \text{BR}(\Pi^{train})}$ must also have low values to encourage no best-response policy of any $\pi^j \in \Pi^{train}$ to be effective for collaborating with a different policy from $\text{BR}(\Pi^{train})$. The incompatibility of a best-response policy for dealing with other policies in $\Pi^{train}$ leads to the need for different collaboration strategies to deal with each policy in $\Pi^{train}$ to emerge.
Based on these two characteristics, we define our diversity metric as:

\[
\text{Div}(\Pi_{\text{train}}, (H^1_t, H^2_t)) = \text{Trace}(C^{\Pi_{\text{train}}, \text{BR}(\Pi_{\text{train}})}(H^1_t, H^2_t)) + \sum_{i,j \in \{1,...,K\}, i \neq j} (C^{\Pi_{\text{train}}, \text{BR}(\Pi_{\text{train}})}^{i,i}(H^1_t, H^2_t) - C^{\Pi_{\text{train}}, \text{BR}(\Pi_{\text{train}})}^{i,j}(H^1_t, H^2_t)) + \sum_{i,j \in \{1,...,K\}, i \neq j} (C^{\Pi_{\text{train}}, \text{BR}(\Pi_{\text{train}})}^{j,i}(H^1_t, H^2_t) - C^{\Pi_{\text{train}}, \text{BR}(\Pi_{\text{train}})}^{j,j}(H^1_t, H^2_t)).
\]

(6.3)

The maximisation of the first term in Equation 6.3 enforces the first characteristic. Meanwhile, the maximisation of the remaining terms encourage a cross-play matrix with low off-diagonal values, which encourages the generated policies fulfil the second desired characteristic previously mentioned.

6.3 MARL-Based Diversity Optimisation

We now describe an optimisation technique utilised by BRDiv to generate \(\Pi_{\text{train}}\) by maximising Equation 6.3. Our proposed optimisation technique is specifically based on the Multi-Agent A2C (MAA2C) algorithm (Papoudakis et al., 2021b). We use the centralised critic of MAA2C to estimate the elements of the cross-play matrix defined in Equation 6.2. Meanwhile, the policies in \(\Pi_{\text{train}}\) alongside their associated best response policies in \(\text{BR}(\Pi_{\text{train}})\) are treated as actors that are trained within our MAA2C approach. A detailed pseudocode of our MARL-based diversity optimisation technique is provided in Algorithm 8 in Section 6.4.

6.3.1 Data Collection

Our MAA2C-based approach separately collects two types of interaction data for training the actors and the centralised critic. First, our approach collects self-play experiences where we let a policy, \(\pi^k \in \Pi_{\text{train}}\), interact with its associated best response policy, \(\pi^{-k} \in \text{BR}(\Pi_{\text{train}})\). The second type of data is cross-play experiences which we collect by letting a policy, \(\pi^j \in \Pi_{\text{train}}\), interact with the best response policy of a different policy, \(\pi^{-k} \in \text{BR}(\Pi_{\text{train}} - \{\pi^j\})\). Both self-play and cross-play interaction data are then stored in separate storage denoted by \(D^{\text{SP}}\) and \(D^{\text{XP}}\) respectively. Note that assuming we also record the identity of the agents generating the experience,
which is \( j \) and \(-k\), each experience stored in the storage is then defined as a 7-tuple, \( \langle (H^1_t, H^2_t), a^j_t, a^{-k}_t, \{ R_t \}, (H^1_{t+1}, H^2_{t+1}), j, -k \rangle \).

### 6.3.2 Actor & Centralised Critic Architecture

As we mentioned at the beginning of Section 6.3, the trained actors in our MAA2C-based approach correspond to the generated teammate policies in \( \Pi^{\text{train}} \) and their associated best response policies. For each \( \pi^i \in (\Pi^{\text{train}} \cup \text{BR}(\Pi^{\text{train}})) \), this policy is represented as a neural network parameterised by \( \theta^i \). We specifically use Long Short-Term Memory (LSTM) networks [Hochreiter and Schmidhuber, 1997] to enable each policy to process a sequence of experiences as input. The computation inside each actor is given by:

\[
\pi(a^i_t|H^i_t; \theta^i) = \text{Softmax}(\text{LSTM}_{\theta^i}(\text{Concatenate}(a^i_t, a^{i-1}_t, c_{t-1}, h_{t-1}))(a^i_t)), \tag{6.4}
\]

with \( c_{t-1} \) and \( h_{t-1} \) being the LSTM’s cell and hidden state that represents the sequence of previous observations, \( H^i_{t-1} \). In the remainder of our description of BRDiv, note that we denote the set of actor parameters from \( \Pi^{\text{train}} \cup \text{BR}(\Pi^{\text{train}}) \) as \( \Theta \).

Similar to the actor networks, the centralised critic used in this optimisation process is also implemented as an LSTM network. Unlike the actor network, the centralised critic network is responsible for estimating elements of the cross-play matrix, \( C^{\Pi^{\text{train}}, \text{BR}(\Pi^{\text{train}})} \), based on Equation 6.2. The computation inside the LSTM network is expressed as:

\[
V^\phi_{i,-j}(H^i_t, H^{-j}_t) = \text{LSTM}_\phi(\text{Concatenate}(o^i_t, o^{i-1}_t, a^{-j}_t, a^{-j-1}_t, i, -j), c_{t-1}, h_{t-1}) \tag{6.5}
\]

assuming \( \phi, c_{t-1} \) and \( h_{t-1} \) denote the LSTM’s parameters, cell and hidden state from processing the sequence of previous observations, \( H^i_{t-1} \) and \( H^{-j}_{t-1} \). In the remainder of this document, note that we drop \( \Pi^{\text{train}} \) as parameters to the cross-play matrix since evaluating each element of this matrix at row \( i \) and column \( j \) does not involve \( \pi^i \) and \( \pi^{-j,*} \). Instead, we evaluate \( V^\phi_{i,-j}(H^i_t, H^{-j}_t) \) by concatenating a one-hot identification of \( i \) and \( -j \) to the centralised critic’s input as indicated by the above equation.

Note that our choice to use LSTM networks to represent the actors and centralised critic of MAA2C is based on the assumption that generated teammate policies are conditioned on observation-action history. In environments where agents have access to the underlying state information, the LSTM networks can instead be replaced with multilayer perceptrons. Furthermore, these architectures can be modified to accommo-
date training in environments with more than two agents by representing each policy in \( \Pi^{\text{train}} \) as a GNN that jointly evaluates the actions of multiple agents at every timestep.

### 6.3.3 Learning Objective

**Centralised critic loss.** The centralised critic network is trained to optimise the \( n \)-step return loss based on a target critic network parameterised by \( \bar{\phi} \). Using the collected self-play, \( D^{SP} \), and cross-play data, \( D^{XP} \), the centralised critic loss function is defined below:

\[
L_{\phi}(D^{SP}, D^{XP}) = \sum_{g \in D^{SP} \cup D^{XP}} \frac{1}{2} \left( V_{i,-j}^{\phi}(H_{t}^{1}, H_{t}^{2}) - \sum_{k=0}^{n-1} \gamma^{k} R_{t+k} - \gamma^{n} V_{i,-j}^{\phi}(H_{t+n}^{1}, H_{t+n}^{2}) \right)^{2}.
\]  

(6.6)

**Actor loss.** We optimise \( \Pi^{\text{train}} \) and their associated best response policies by minimising the actor loss function defined below:

\[
L_{\theta}(D^{SP}, D^{XP}) = \sum_{g \in D^{SP} \cup D^{XP}} \left( -\log \left( \pi(a_{t}^{i} | H_{t}^{1} ; \theta_{t}) \pi(a_{t}^{j} | H_{t}^{2} ; \theta_{t}) \right) \right)
\]

\[
A_{i,-j}^{\phi}(H_{t}^{1}, H_{t}^{2}, \{ R_{t+k} \}_{k=0}^{n-1}, H_{t+n}^{1}, H_{t+n}^{2})
\]

where \( A_{i,-j}^{\phi}(H_{t}^{1}, H_{t}^{2}, \{ R_{t+k} \}_{k=0}^{n-1}, H_{t+n}^{1}, H_{t+n}^{2}) \) is an advantage function that we define as:

\[
\text{Div}(C_{i,-j}^{\text{pred}, \phi}(H_{t}^{1}, H_{t}^{2}, \{ R_{t+k} \}_{k=0}^{n-1}, H_{t+n}^{1}, H_{t+n}^{2})) - \text{Div}(C_{\text{base}}^{\phi}(s_{t})).
\]  

(6.8)

Based on the diversity metric defined in Equation 6.3, this advantage function evaluates the difference of the return-based diversity metric associated to two matrices whose elements are defined below:

\[
C_{i,-j,p,q}^{\text{pred}, \phi}(H_{t}^{1}, H_{t}^{2}, \{ R_{t+k} \}_{k=0}^{n-1}, H_{t+n}^{1}, H_{t+n}^{2}) = \begin{cases} 
V_{p,-q}^{\text{pred}, \phi}(I), & \text{if } (p, q) = (i, j) \\
V_{p,-q}^{\phi}(H_{t}^{1}, H_{t}^{2}), & \text{otherwise}
\end{cases}
\]

(6.9)

\[
C_{m,n}^{\text{base}, \phi}(H_{t}^{1}, H_{t}^{2}) = V_{m,-n}^{\phi}(H_{t}^{1}, H_{t}^{2}),
\]

(6.10)

with,

\[
I = \langle \{ R_{t+k} \}_{k=0}^{n-1}, H_{t+n}^{1}, H_{t+n}^{2} \rangle,
\]

(6.11)

and,

\[
V_{p,-q}^{\text{pred}, \phi}(\{ R_{t+k} \}_{k=0}^{n-1}, H_{t+n}^{1}, H_{t+n}^{2}) = \sum_{k=0}^{n-1} \gamma^{k} R_{t+k} + \gamma^{n} V_{i,-j}^{\phi}(H_{t+n}^{1}, H_{t+n}^{2}).
\]

(6.12)
In the above equation, $C_{i-j}^{\text{pred}}(H_1^t, H_2^t, \{R_{t+k}\}_{k=0}^{n-1}, H_1^{t+n}, H_2^{t+n})$ is a matrix whose elements are computed based on the previously defined critic except for one element corresponding to the predicted returns from the collaboration between agent $i$ and $-j$, whose interaction generated the data used for training. We set its value for this replaced matrix element to the predicted $n$-step returns computed based on the gathered experience. We use $n$-step return-based replacement to reduce the bias of gradients associated with the actor loss updates, which is also a commonly used trick in single-agent actor-critic methods. Meanwhile, $C_{\text{base}}^{\phi}(s_t)$ is a baseline cross-play matrix whose elements only depend on $H_1^t$ and $H_2^t$. We include a baseline function in our actor loss to reduce the variance of the gradient updates.

6.4 BRDiv Pseudocode

We complete the description of our method by providing a pseudocode for the teammate generation process undergone in BRDiv, shown in Algorithm 8. An important part of Algorithm 8 is a call to the COMPUTE LOSS function that evaluates the loss functions minimised by BRDiv. How BRDiv utilises the gathered self-play and cross-play experience to compute the minimised loss functions is described in Algorithm 9.

6.5 Teammate Generation Experiments

We present the experiments we conduct to demonstrate the effectiveness of BRDiv in improving the robustness of an AHT learner when dealing with previously unseen teammate types. We provide details of the environment used in our teammate generation experiments in Section 6.5.1. This is followed by an overview of our experiments’ AHT training and evaluation process in Section 6.5.2. Section 6.5.3 then details the baseline approaches we compare BRDiv against. We then present and analyse the results of the teammate generation experiments in Section 6.5.4. Finally, Section 6.5.5 ends this chapter by describing the behaviours of teammate types generated by BRDiv and other compared baselines.

6.5.1 Environment

We evaluate BRDiv and baseline approaches in the Cooperative Reaching environment which is suitable for teammate generation experiments following the existence of mul-
Algorithm 8 BRDiv Teammate Generation Process

Input:
Number of training episodes, \( n_{\text{eps}} \).
Episode length, \( T \).
Update period, \( t_{\text{update}} \).
Number of generated teammate types, \( K \).
Initial population actor network parameters, \( \Theta = \{\theta_1, \theta_2, \ldots, \theta_K\} \).
Initial centralised critic parameters, \( \phi \).
Target centralised critic parameters, \( \bar{\phi} \).
Learning Rate, \( \alpha \).
Target network update coefficient, \( \bar{\alpha} \).
Environment for SP and XP interaction, \( \text{env}^{\text{SP}} \) & \( \text{env}^{\text{XP}} \).

1: for \( i = 1 \) to \( n_{\text{eps}} \) do
2: \( t \leftarrow 0 \)
3: \( \mathcal{D}^{\text{SP}}, \mathcal{D}^{\text{XP}} \leftarrow \{\}, \{\} \)
4: \( \text{ID}^{\text{SP}} \sim \text{Uniform}\{1, \ldots, K\} \quad \triangleright \text{Sample Population ID for SP} \)
5: \( \text{ID}^{\text{XP}}, \text{ID}^{\text{XP}}_2 \sim \text{Uniform}\{(i, j) | i, j \in 1, \ldots, K, i \neq j\} \quad \triangleright \text{Sample Population ID for XP} \)
6: Observe \( H^{\text{SP}}_0 = (o^{1, \text{SP}}_0, o^{2, \text{SP}}_0) \) and \( H^{\text{XP}}_0 = (o^{1, \text{XP}}_0, o^{2, \text{XP}}_0) \) from \( \text{env}^{\text{SP}} \) and \( \text{env}^{\text{XP}} \) respectively.
7: \( H^{1, \text{SP}}_0, H^{2, \text{SP}}_0, H^{1, \text{XP}}_0, H^{2, \text{XP}}_0 \leftarrow \{o^{1, \text{SP}}_0\}, \{o^{2, \text{SP}}_0\}, \{o^{1, \text{XP}}_0\}, \{o^{2, \text{XP}}_0\} \)
8: while \( t < T \) do
9: // Self-Play Data Collection
10: \( a^{1, \text{SP}}_t \sim \pi\left(a^{1, \text{SP}}_t | H^{1, \text{SP}}_t, \text{ID}^{\text{SP}}_t; \Theta^{\text{SP}}\right) \) and \( a^{2, \text{SP}}_t \sim \pi\left(a^{2, \text{SP}}_t | H^{2, \text{SP}}_t, \text{ID}^{\text{SP}}_t; \Theta^{\text{SP}}\right) \)
11: \( r^{\text{SP}}_{t+1}, H^{\text{SP}}_{t+1} \leftarrow \text{env}^{\text{SP}}\left(H^{\text{SP}}_t, a^{\text{SP}}_t\right) \)
12: \( \mathcal{D}^{\text{SP}} \leftarrow \mathcal{D}^{\text{SP}} \cup \{H^{\text{SP}}_t, a^{\text{SP}}_t, r^{\text{SP}}_{t+1}, H^{\text{SP}}_{t+1}, \text{ID}^{\text{SP}}_t\} \)
13: // Cross-Play Data Collection
14: \( a^{1, \text{XP}}_t \sim \pi\left(a^{1, \text{XP}}_t | H^{1, \text{XP}}_t, \text{ID}^{\text{XP}}_t; \Theta^{\text{XP}}\right) \) and \( a^{2, \text{XP}}_t \sim \pi\left(a^{2, \text{XP}}_t | H^{2, \text{XP}}_t, \text{ID}^{\text{XP}}_t; \Theta^{\text{XP}}\right) \)
15: \( r^{\text{XP}}_{t+1}, H^{\text{XP}}_{t+1} \leftarrow \text{env}^{\text{XP}}\left(H^{\text{XP}}_t, a^{\text{XP}}_t\right) \)
16: \( \mathcal{D}^{\text{XP}} \leftarrow \mathcal{D}^{\text{XP}} \cup \{H^{\text{XP}}_t, a^{\text{XP}}_t, r^{\text{XP}}_{t+1}, H^{\text{XP}}_{t+1}, \text{ID}^{\text{XP}}_t, \text{ID}^{\text{XP}}_2\} \)
17: if \( t \mod t_{\text{update}} = 0 \) then
18: // Parameter Update
19: \( \mathcal{L}_{\Theta, \phi}(\mathcal{D}^{\text{SP}}, \mathcal{D}^{\text{XP}}) \leftarrow \text{COMPUTE_LOSS}(\mathcal{D}^{\text{SP}}, \mathcal{D}^{\text{XP}}, \Theta, \phi, \bar{\phi}) \)
20: for \( \theta_i \in \Theta \) do
21: \( \theta_i \leftarrow \text{GRADIENT_DESCENT}(\theta_i, \alpha, \nabla_{\theta_i} \mathcal{L}_{\Theta, \phi}(\mathcal{D}^{\text{SP}}, \mathcal{D}^{\text{XP}})) \)
22: end for
Chapter 6. Generating Teammates For Ad Hoc Teamwork

Multiple cooperation strategies that can be utilised by agents for effective collaboration. Cooperative reaching is a simple environment situated in a 5 × 5 grid world. In this grid world, two agents’ actions allow them to move along the four cardinal directions. The goal of the agents is to reach and jointly stay in a grid whose location belongs to the set of reward-providing coordinates, \( F = \{(0,0), (0,4), (4,0), (4,4)\} \). Within these reward-providing coordinates, \((0,0)\) and \((4,4)\) provide a reward of 1 to both agents once they are in the grid with this coordinate. Meanwhile, the grid in \((0,4)\) and \((4,0)\) only provide a reward of 0.75 once both agents arrived. In this environment, the collaboration strategies correspond to the distinct ways a teammate may select a destination grid within \( F \). A robust AHT learner should ideally learn to follow its teammates towards any reward-providing coordinates.

Note that our experiments in this chapter are not executed in the environments we used for our open AHT experiments since our teammate generation method is currently limited to closed two-player open ad hoc teamwork problems. Furthermore, we focus on environments where the need for adaptive agents results from the different strategic conventions that teammates adopt to obtain high returns in the environment. This contrasts with the environments for our open AHT problem where adaptive agents arise from having to deal with teams of different sizes.

6.5.2 Teammate Generation & Evaluation Process

Our methods can be divided into two crucial stages. In the first stage, we run BR-Div and other baseline teammate generation methods to create training teammates for training a learner. The second stage utilises the resulting teammates from the first stage to train a learner, which returns when collaborating with a previously unseen teammate.
Algorithm 9 Loss Computation

**Input:**
- Self-play and cross-play data, \( \mathcal{D}^{SP} \) & \( \mathcal{D}^{XP} \).
- Population actor network parameters, \( \Theta \).
- Centralised critic parameters, \( \phi \).
- Target centralised critic parameters, \( \bar{\phi} \).

1: function \text{COMPUTE\_LOSS}(\mathcal{D}^{SP}, \mathcal{D}^{XP}, \Theta, \phi, \bar{\phi})
2: \( t_{\text{start}} \leftarrow \) first time in the buffers \( \mathcal{D}^{SP}, \mathcal{D}^{XP} \)
3: \( t_{\text{end}} \leftarrow \) latest time in the buffers \( \mathcal{D}^{SP}, \mathcal{D}^{XP} \)
4: \( V_{\text{target}} \leftarrow V(H_{t_{\text{end}}+1}, ID^{SP}, ID^{SP}; \bar{\phi}) \)
5: \( \mathcal{L}_{\phi}^{SP} \leftarrow 0 \quad \triangleright \text{Compute Self-Play Critic Loss} \)
6: for \( t = t_{\text{end}} \) to \( t_{\text{start}} \) do
7: \( V_{\text{pred}} \leftarrow V(H_t^{SP}, ID^{SP}, ID^{SP}; \phi) \)
8: \( V_{\text{target}} \leftarrow \begin{cases} R_{t}^{SP} & \text{if episode terminates at } t \\ R_{t}^{SP} + \gamma V_{\text{target}} & \text{otherwise.} \end{cases} \)
9: \( \mathcal{L}_{\phi}^{SP} \leftarrow \mathcal{L}_{\phi}^{SP} + \frac{1}{2} (V_{\text{pred}} - V_{\text{target}})^2 \)
10: end for
11: \( V_{\text{target}} \leftarrow V(H_{t_{\text{end}}+1}^{XP}, ID_{1}^{XP}, ID_{2}^{XP}; \tilde{\phi}) \)
12: \( \mathcal{L}_{\phi}^{XP} \leftarrow 0 \quad \triangleright \text{Compute Cross-Play Critic Loss} \)
13: for \( t = t_{\text{end}} \) to \( t_{\text{start}} \) do
14: \( V_{\text{pred}} \leftarrow V(H_t^{XP}, ID_{1}^{XP}, ID_{2}^{XP}; \phi) \)
15: \( V_{\text{target}} \leftarrow \begin{cases} R_{t}^{XP} & \text{if episode terminates at } t \\ R_{t}^{XP} + \gamma V_{\text{target}} & \text{otherwise.} \end{cases} \)
16: \( \mathcal{L}_{\phi}^{XP} \leftarrow \mathcal{L}_{\phi}^{XP} + \frac{1}{2} (V_{\text{pred}} - V_{\text{target}})^2 \)
17: end for
18: \( V_{\text{bootstrap}} \leftarrow V(H_{t_{\text{end}}+1}^{SP}, ID^{SP}, ID^{SP}; \phi) \)
19: \( \mathcal{L}_{\Theta}^{SP} \leftarrow 0 \quad \triangleright \text{Compute Self-Play Actor Loss} \)
20: for \( t = t_{\text{end}} \) to \( t_{\text{start}} \) do
21: \( M_{\text{baseline}} \leftarrow \text{TO\_XP\_MATRIX}(\{V(H_{t}^{SP}, i, j; \phi)|i, j \in 1, \ldots, N\}) \)
22: \( V_{\text{bootstrap}} \leftarrow \begin{cases} R_{t}^{SP} & \text{if episode terminates at } t \\ R_{t}^{SP} + \gamma V_{\text{bootstrap}} & \text{otherwise.} \end{cases} \)
23: \( M_{\text{pred}} \leftarrow M_{\text{baseline}} \)
24: \( M_{\text{pred,ID}^{SP}, ID^{SP}} \leftarrow V_{\text{bootstrap}} \quad \triangleright \text{Replace matrix element of interacting populations} \)
Chapter 6. Generating Teammates For Ad Hoc Teamwork

25: \[ L^{\text{SP}}_\Theta \leftarrow L^{\text{SP}}_\Theta - \log(\pi(a^1_1, \Theta^{1, \text{SP}}_I | H^1, \Theta^{1, \text{SP}}_I) \pi(a^1_2, \Theta^{1, \text{SP}}_I | H^2, \Theta^{1, \text{SP}}_I))(\text{Div}(M_{\text{pred}}) - \text{Div}(M_{\text{baseline}})) \]

26: \[ \text{end for} \]

27: \[ V_{\text{bootstrap}} \leftarrow V(H^{\text{XP}}_{t_{\text{end}}+1}, \text{ID}^{\text{XP}}_1, \text{ID}^{\text{XP}}_2, \phi) \]

28: \[ L^{\text{XP}}_\Theta \leftarrow 0 \quad \triangleright \text{Compute Cross-Play Actor Loss} \]

29: \[ \text{for } t = t_{\text{end}} \text{ to } t_{\text{start}} \text{ do} \]

30: \[ M_{\text{baseline}} \leftarrow \text{TO \_XP \_MATRIX} \left( \{ V(H^{\text{XP}}_t, i, j; \phi) \}_{i, j = 1, \ldots, N} \right) \]

31: \[ V_{\text{bootstrap}} \leftarrow \begin{cases} i^{\text{XP}}_t, & \text{if episode terminates at } t \\ i^{\text{XP}}_t + \gamma V_{\text{bootstrap}}, & \text{otherwise.} \end{cases} \]

32: \[ M_{\text{pred}} \leftarrow M_{\text{baseline}} \]

33: \[ M_{\text{pred,ID}^{\text{XP}}_1, \text{ID}^{\text{XP}}_2} \leftarrow V_{\text{bootstrap}} \quad \triangleright \text{Replace matrix element of interacting populations} \]

34: \[ L^{\text{SP}}_\Theta \leftarrow L^{\text{SP}}_\Theta - \log(\pi(a^1_1, H^{\text{XP}}_t, \text{ID}^{\text{XP}}_1; \Theta^{1, \text{XP}}_I) \pi(a^1_2, H^{\text{XP}}_t, \text{ID}^{\text{XP}}_2; \Theta^{1, \text{XP}}_I))(\text{Div}(M_{\text{pred}}) - \text{Div}(M_{\text{baseline}})) \]

35: \[ \text{end for} \]

36: \[ \text{Return: } L^{\text{SP}}_\Phi + L^{\text{XP}}_\Phi + L^{\text{SP}}_\Theta + L^{\text{XP}}_\Theta \]

37: \[ \text{end function} \]

Type in training is periodically evaluated.

For each evaluated teammate generation method, the first stage of our experiments runs each method for five different seeds to produce \( K \) teammate types for AHT training. The teammate generation method learns for \( T \) total timesteps within each seed. \( K \) and \( T \) generally differed between environments. Each teammate generation method is trained for 16 million timesteps to produce four teammate types due to the simplicity of the environment. By the end of this learning process, we save the collection of training teammate types, \( \Pi_{\text{train}} \), produced by executing an evaluated algorithm at each seed.

\( \Pi_{\text{train}} \) is then used by an AHT method to train a learner. Our evaluation process ensures fair evaluation using the same AHT algorithm to train a learner based on each teammate generation method. We use the Local Information Agent Modelling (LIAM) (Papoudakis et al., 2021a) as our main AHT algorithm. Although GPL is another potential AHT method for training the learner, the fixed number of teammates in the environments used in our experiments provides less justification for using GPL for training.

For each generated \( \Pi_{\text{train}} \) using different seeds, we use LIAM to separately train
We specifically train LIAM in Cooperative Reaching 16 million timesteps. At the beginning of each episode, we uniformly sample a teammate type from the set of generated teammates defined in Appendix A. At different checkpoints during AHT training, we then record the learner’s returns when interacting with holdout teammate types not seen during training for each seed.

We report the mean returns of each method at every checkpoint as a measure of the produced AHT learner’s robustness. Results from the different seeds also enable us to establish a 95% confidence interval over the robustness of AHT learners produced by each method, which allows us to argue over the significance of the difference in robustness between teammate generation methods. The resulting returns of a learner trained through generated teammate types produced by BRDiv and baseline approaches are reported and analysed in Section 6.5.4.

6.5.3 Baseline Methods

Our experiments compared BRDiv with two types of baselines. The first type of baseline comprises previous methods for automatically generating teammates in AHT or related problems, such as zero-shot coordination. Meanwhile, the second type of baseline consists of the ablation of BRDiv, which removes parts of BRDiv responsible for encouraging ineffective collaboration between a generated teammate policy and the best response policy associated with another generated teammate type. The details of these methods and their implementation based on our code is provided below:

Prior teammate generation methods. Among methods under this category, we choose TrajeDi ([Lupu et al., 2021]) as a representative baseline. We choose TrajeDi following its usage of the action discounting term, which provides additional flexibility when defining the optimised information-theoretic diversity metric. Prior teammate generation methods other than TrajeDi typically define their optimised diversity metric in terms of an agent’s overall trajectory or its selected action at each timestep, which both have their drawbacks. TrajeDi’s action discounting term enables users to tune the resemblance of its optimised diversity metric to an action diversity and trajectory diversity-based approach.

Implementing TrajeDi based on our pseudocode is also straightforward. First, we remove all loss functions evaluated based on cross-play data since TrajeDi only relies on self-play data for training. This is effectively done by subtracting $L^{XP}_\phi$ and $L^{XP}_\Theta$ from the loss function returned by Algorithm 9. Using $D^{SP}$ for its evaluation, we also
add the negative of the information-theoretic diversity metric maximised by TrajeDi to the loss function returned by Algorithm 9. These straightforward changes enable us to generate teammates with TrajeDi using our proposed pseudocode.

**Ablations of BRDiv.** We also compare BRDiv against an ablation which independently trains $K$ teammate policies with MAA2C (Papoudakis et al., 2021b) without maximising BRDiv’s proposed diversity metric outlined in Equation 6.3. Comparing BRDiv’s resulting performance against this ablation helps us identify the impact of optimising our proposed diversity metric on the resulting learner’s robustness when dealing with previously unseen teammates. Similar to our TrajeDi baseline, we implement this ablation with our pseudocode by first subtracting $L^\phi$ and $L^\Theta$ from BRDiv’s original loss function. Furthermore, when computing MAA2C’s actor loss, we use a modified diversity metric that subtracts all terms on the right-hand side of Equation 6.3 except for the first term. This modification ensures that all $K$ generated policies are only trained to maximise their performance when collaborating with its best response policy.

### 6.5.4 Results in Collaboration With Novel Teammates

This section provides the results of our experiments when using the generated teammates to train a learner. Figure 6.3 show the performance of LIAM’s resulting policy at different checkpoints when collaborating against holdout teammate types and teammate types generated by each evaluated method. While the performance of LIAM against the generated set of teammates cannot be compared between different methods since they are produced by training against different sets of teammates, we see that we have sufficiently trained LIAM until its performance against the generated teammates have plateaued.

Figure 6.3 also indicates that BRDiv produces a more robust learner that achieves higher average returns than other evaluated methods. BRDiv achieves significantly better returns than other evaluated methods because each generated teammate managed to find a policy that makes the teammate move towards different reward-producing coordinates in the environment. Meanwhile, other methods do not enable the set of generated teammates to cover the entire set of reward-producing coordinates. When interacting with the holdout set of teammate types, the learner produced by training against BRDiv’s generated teammates is then able to robustly follow any holdout teammate types even if they are heading towards suboptimal reward-producing coordinates. By
6.5. Teammate Generation Experiments

Figure 6.3: **Ad hoc teamwork results.** For each evaluated teammate generation method, we record the average and 95% confidence bounds of the returns achieved by LIAM during collaboration with the (a) generated and (b) hold out teammate types. These confidence bounds are computed by utilising 5 seeds. The visualisation shows that we have trained the learner to the point its performance converges.

In contrast, the absence of teammate policies that move towards a subset of the reward-producing coordinates prevents other evaluated teammate generation methods from producing a learner capable of moving towards these unreachable coordinates.

### 6.5.5 Generated Teammate Behaviours

This section provides two additional pieces of empirical evidence regarding the effectiveness of BRDiv to generate teammates for AHT training. First, the cross-play matrices displayed in Figure 6.4 further demonstrate that BRDiv achieves its main objective of finding policies that require different best-response policies for effective collaboration. Specifically, all four policies generated via BRDiv require different best-response policies. Unlike BRDiv, TrajeDi and the independent baseline did not discover a cross-play matrix that achieves the optimal diversity metric defined in Equation 6.3. However, note that TrajeDi is slightly better than the independent baseline by discovering two different clusters of teammate policies that requires different best response policies.

Our second piece of empirical evidence of the effectiveness of BRDiv is provided in Figure 6.5. This Figure displays the trajectories exhibited by the collections of generated policies whose cross-play matrix we displayed in Figure 6.4. As we have briefly alluded to in Section 6.5.4, teammates generated by BRDiv move towards different reward-providing coordinates in the environment including the suboptimal ones.
Figure 6.4: **Cross-play matrices in Cooperative Reaching.** From the five collection of teammate types obtained by running each method using five different seeds, we display the cross-play matrix from the interaction between agent policies produced by a with maximum diversity according to Equation 6.3.

Meanwhile, teammates generated by TrajeDi and the independent baseline only move towards coordinates providing the optimal reward.

Figure 6.5 also provides an example of superficial behavioural differences that BRDiv aims to avoid. The teammate policies achieved by the independent baseline and TrajeDi display superficial differences since a policy that moves towards the optimal reward-providing coordinates can effectively deal with all the generated policies. Meanwhile, BRDiv’s generated policies all require distinct best-response policies that require the learner to follow a teammate towards different directions. This provides evidence of the reason behind BRDiv’s robust learners explained in Section 6.5.4.
Figure 6.5: Generated policy visualisations for Cooperative Reaching. We display example trajectories displayed by the generated teammates for a single run of the evaluated algorithms. In the above visualisation, the generated agent is symbolised by the red circle while the trajectory of each of the four teammate policies generated by a method is visualised by an arrow with a different colour. With BRDiv, each generated policy learns to go towards different reward-providing coordinates in the environment. Each teammate generated by independently training each generated policy using the same MARL method produces teammates that cover both of the optimal reward-providing coordinates. Meanwhile, TrajeDi produces teammates that go towards the optimal reward-providing coordinates using different trajectories.

6.6 Chapter Summary

In this chapter, we discussed the importance of generating a collection of training teammate policies, $\Pi^{\text{train}}$, that require different best-response policies to improve the robustness of an AHT agent. To achieve this, we proposed a teammate generation method that optimises BRDiv, a diversity metric designed to prevent the emergence of superficial differences between policies in $\Pi^{\text{train}}$. Based on a comparison against TrajeDi and a baseline that independently trains different teammate policies via MARL, we demonstrate that optimising BRDiv achieves higher average returns when dealing with near-optimal previously unseen teammate policies.

Further analysis of the generated teammates’ behaviour shows that optimising BRDiv avoids generating teammates with superficial differences. At the same time, $\Pi^{\text{train}}$ generated by optimising BRDiv covers a comprehensive set of reward-maximising teammate behaviours. Training against this set of teammates eventually produced teammates that can perform a broader range of strategies to collaborate against previously unseen teammate policies.
Chapter 7

Conclusion & Future Work

This thesis addresses the problem of ad hoc teamwork (AHT), which aims to design a learner that can collaborate with unknown teammates without prior coordination mechanisms. Prior methods typically achieve the goal of AHT by relying on two assumptions that do not hold in many potential real-world applications. First, it is commonly assumed that a learner operates in a closed environment where the set of teammates interacting with a learner remains the same throughout an interaction. Second, prior AHT methods also assume access to a set of manually designed teammates, whose interactions with the learner provide the necessary experiences to train a learner capable of effective collaboration with unknown teammates. Since these assumptions are often difficult to fulfil, they impede the applicability of existing AHT methods towards a broader range of application domains.

The previous chapters in this thesis present methods that enable the design of a learner even when the aforementioned assumptions are not upheld. Chapter 3 starts by formally defining the open AHT as an open stochastic bayesian game (OSBG), which addresses the inability of previous AHT formulations to model environment openness. In Chapter 4, we introduced the Graph-based Policy Learning (GPL) algorithm to train a learner that effectively collaborates in open environments where its set of teammates and their associated types may change throughout an interaction. Chapter 5 then extends GPL to train a learner in open ad hoc teamwork problems under partial observability. Finally, Chapter 6 introduces BRDiv, a MARL-based teammate generation method to train teammate policies that can be used to train a robust learner.

In the remaining sections of this chapter, we conclude this thesis by reiterating our findings and discussing potential limitations and future research directions for our proposed methods. Section 7.1 summarises the important findings from the experiments
presented in Chapters 4, 5, and 6. We then discuss the potential limitations of our proposed methods in Section 7.2. Finally, Section 7.3 points towards promising research directions to address the issues we have identified regarding our proposed methods.

7.1 Key Results

Following the experiments we have conducted for each method proposed in this thesis, we highlight significant results that explain the effectiveness of our proposed methods to solve the open problems outlined in Section 1.2. Note that a brief summary of these results has also been provided in Section 1.3 and the beginning of each chapter that introduces our proposed methods. A summary of our key results is provided below:

- **(Section 4.9.2 in Chapter 4) GNNs for open AHT.** We empirically demonstrate the importance of Graph Neural Networks (GNNs) for open ad hoc teamwork by contrasting the performance of GPL, the evaluated GNN-based single-agent RL methods, and the remaining baselines. In both the training process and generalisation experiments, GNN-based methods significantly outperform single-agent RL baselines that do not use GNNs for action-value function estimation. GNN-based methods’ significantly higher returns in these experiments follow from GNNs being a better neural network architecture for handling inputs with variable sizes, which aligns with previous works that compare GNNs against alternative neural network architectures in other learning problems (Hamilton et al., 2017; Jiang et al., 2019; Huang et al., 2020). Consequently, GNNs are a prime candidate model of choice for future methods addressing open ad hoc teamwork.

- **(Sections 4.9.2, 4.9.3, and 4.9.4 in Chapter 4) Joint action value estimation for open AHT.** Comparisons between the returns resulting from GPL and other GNN-based single-agent RL methods highlight the importance of GPL’s action-value estimation method for open ad hoc teamwork. Even in level-based foraging where all these methods produce similar returns during training, GPL improves the generalisation performance of the learner by producing significantly higher returns than our GNN-based baselines when applied to a previously unseen open process. We credit this improved performance to GPL’s estimation of the effects of teammates’ actions towards the learner’s returns using its joint action value model. Based on an experiment conducted in FortAttack, we show
in Section 4.9.3 that GPL’s joint action value model enables a learner to implicitly learn useful skills displayed by a better-performing teammate. Furthermore, Section 4.9.4 also demonstrates that single-agent RL methods not equipped with a joint action value model cannot learn useful skills from teammates as done in GPL. Therefore, this indicates the importance of joint action value models as components for solutions to open ad hoc teamwork.

• (Sections 5.8.2 and 5.8.3 in Chapter 5) In Chapter 5, we compare the returns from different latent variable inference methods when being applied to open ad hoc teamwork under partial observability. Our results did not show any latent variable inference method consistently outperforming other methods in all environments. For the learner to produce high returns, we observe that different environments require a specific focus on inferring a subset of the set of latent variables. By measuring the prediction error resulting from reconstructing a latent variable’s real value based on our models’ inferred representations, we also discover that different latent variable inference models specialise in representing different subsets of the latent variables. These findings highlight the importance of domain knowledge regarding the important set of latent variables that must be inferred to perform well in a certain task. Based on this knowledge, a suitable latent variable inference model can be designed to provide high returns in a task. For tasks where latent state and teammate joint action inference is most important to deliver higher returns, we recommend using the autoencoder and variational autoencoder-based methods. Meanwhile, tasks requiring highly accurate teammate existence prediction should use particle filter-based belief inference methods instead.

• (Sections 6.5.4 and 6.5.5 in Chapter 6) We demonstrate the importance of generating teammate policies that require different best-response policies to improve the robustness of an AHT agent. Based on a comparison against TrajeDi (Lupu et al., 2021) and a baseline that independently discovers different teammate policies via MARL, our experiments in the cooperative reaching environment show that our proposed method achieves significantly higher returns when dealing with previously unseen teammate types. Further analysis of the generated teammates’ behaviour shows that our proposed method can cover a more comprehensive set of reward-maximising teammate behaviours compared to the baselines. Training against this set of teammates produced teammates that can perform
Chapter 7. Conclusion & Future Work

7.2 Limitations of Proposed Methods

7.2.1 GPL & Latent Variable Inference Models

As we discussed in our key findings on GPL in Section 7.1, GPL’s utilisation of GNNs enables it to yield higher returns than baselines in open AHT. Furthermore, we also identified the joint action value model as a critical component for decision-making in open AHT. Despite this, we must recognise the potential issues resulting from the design and usage of GPL’s joint action value model.

The first issue concerns GPL’s usage of Coordination Graphs (CGs) as its joint action value model. As highlighted in Equation 4.5, CGs assume that the joint action value function can be factorised into a sum of singular and pairwise utility terms. However, even in matrix games, it is relatively straightforward to find example joint action value functions that cannot be accurately factorised according to this assumption. Forcing the usage of CGs in environments where CGs’ assumed value factorisation is not upheld will result in biased joint action value estimates. Eventually, this will produce biased action-value function estimates which may significantly impact the learner’s returns when solving the AHT problem.

The second issue pertains GPL’s assumption that each agent’s action space is discrete. Under discrete action spaces, marginalising teammates’ joint actions to compute GPL’s action-value function is straightforward. This becomes challenging when we deal with continuous spaces since the summations in Equation 4.10 have to be replaced with an integral over the actions. When the joint action value and agent model are represented as neural networks, the integral over the action likelihood may not have a closed-form expression. This prevents an exact evaluation of Equation 4.10 which results in the need to approximate the value of this equation. However, approximations to the action-value function can potentially introduce additional errors that may impact the returns produced by the learner.

Another issue relates to a teammate’s fixed type during its time in the open AHT environments used by our experiments for GPL. In many real-world scenarios, a learner has to interact with teammates that adapt their policies according to the learner’s behaviour. While GPL’s type inference model can still infer representations for such
adaptive teammates, the representations may not help with adaptive teammates since GPL was only trained to deal with teammates with fixed types. Additionally, the type inference network does not have a particular design that makes it tailored for modelling adaptive teammates either.

Finally, the last issue relates to GPL’s potential performance when dealing with teams containing teammate types not encountered during training. While GPL’s neural network-based models potentially enable it to generalise its action-value function estimates to an interaction against novel teammates, the action-value function estimate may not be accurate for these scenarios. This eventually may negatively impact the returns yielded by GPL when collaborating with teammates containing previously unseen types.

Our proposed methods to extend GPL to partially observable environments will also display the previously mentioned limitations of GPL simply because of their usage of GPL. That aside, the key findings provided in Section 7.1 regarding the performance of our proposed latent variable inference methods suggest a potential for improvements. Specifically, our results suggest the need for a latent variable inference model that can accurately infer all important latent variables for decision-making, such as teammates’ types, existence, selected actions, and state features. Having such a model potentially leads towards the design of a learner that performs better in open AHT under partial observability.

7.2.2 BRDiv

Although our results in the teammate generation experiments show that BRDiv can generate teammate policies that require different strategies for effective collaboration, we note that this is not the only type of diversity displayed by decision-making agents in real-world problems. In many applications of AHT, a learner also has to deal with teammates that vary in their ability to maximise the teams’ returns. For example, even with different teammates that prefer a specific role such as being a striker, we see a wide range of skill levels between potential teammates in a pick-up soccer game. A teammate’s ability may range from having the skills of an amateur player to possessing elite skills displayed by top-division professional players. Currently, this diversity is not something that can be discovered by BRDiv. The first term on the right-hand side of Equation 6.3 encourages BRDiv to generate teammates with near-optimal policies. By only training a learner against teammates generated by BRDiv, this limitation po-
tentially results in a learner yielding suboptimal returns when dealing with teammates with a low skill level.

BRDiv is also currently limited towards two-player problems where we only need to generate a single teammate. In many real-world problems such as those addressed in open ad hoc teamwork, generating a team of agents of different types is desirable. While BRDiv can be modified to generate a policy by training a collection of actors for each generated team, potential challenges arise when generating more than one teammate. Specifically, an increase in the number of agents results in an exponential increase in the possible team configurations. Training a robust learner based on generated teams will then require BRDiv to generate a more significant number of actors via deep multi-agent RL. Since training agents via MARL may require millions of experiences in many realistic applications, training this large collection of actors potentially requires significant computational resources that may not be available to all AHT practitioners.

Finally, further experiments that evaluate BRDiv in more complex environments are required to demonstrate its applicability in a broader range of AHT problems. In particular, experiments in environments with a larger state space can further justify BRDiv’s use of function approximations that require a hefty amount of resources to train such as neural networks. To highlight the motivation behind the need for a teammate generation method such as BRDiv, ideally an environment where a set of good teammates are difficult to manually engineer can also be useful.

7.3 Future Research Directions

Based on the previously mentioned limitations of our proposed method, we outline a few research directions that may further improve our proposed methods. These promising research directions are provided below:

**Coordination hypergraphs & structure learning.** GPL’s limitation from using CGs for its joint action value model can be addressed using models that employ more complex factorisations than CGs. This can be made possible by factorising the learner’s joint action value function into singular and pairwise utility terms and including terms involving more than two agents. This introduction of utility terms involving more than two agents effectively extends coordination graphs into hypergraphs [Berge, 1985], where agents involved in a utility term are connected by an hyperedge which can connect more than two agents. Despite this potential, no work has explored the
extension of coordination graphs into hypergraphs.

Note that introducing more complex terms means that the action-value function will no longer be defined in singular and pairwise terms as in Equation 4.10. Instead, the resulting action-value function will involve terms with more than two agents’ actions as input. The evaluation of utility terms with more than two agents can be computationally expensive due to the exponential increase in the set of joint actions to be evaluated. Therefore, the usage of coordination hypergraphs must be combined with additional techniques that learn the simplest coordination hypergraph structure that still provides reasonably accurate estimates of the joint action-value function. While such techniques are not yet available for coordination hypergraphs, promising techniques for graph structure learning have been introduced for CGs by Yang et al. (2022). Both coordination hypergraphs and this technique provide potential research directions to alleviate the limitations of GPL’s restrictive joint action value model expressiveness.

Monte Carlo-based action-value estimates. Following its effectiveness in partially observable open AHT, a potential solution to estimate GPL’s action-value function under a continuous action space is to use a Monte Carlo approach. The application of a Monte Carlo approach is possible because a learner’s action-value function is its expected joint action-value function under the distribution of teammates’ actions (Equation 4.2). Specifically, we can sample multiple actions according to the action distribution outputted by GPL’s agent model. The sampled actions can then be provided as input to the joint action-value function, which computes the action-value function as the average joint action-value function over the generated action samples. Nevertheless, obtaining an accurate estimate of the action-value function will be increasingly challenging as the number of agents increases. This potentially requires us to sample more agent actions as the number of agents increases.

Modelling teammates with adaptive policies. A potential solution to address GPL’s limitation when dealing with adaptive teammates is to use methods that infer how teammates adapt their policies according to a learner’s actions (Xie et al., 2021). Xie et al. (2021) specifically proposed a neural network model that infers a teammate representation that is informative of their strategy and their adaptation methods. An exciting research direction is incorporating this model in GPL by using it to replace GPL’s type inference model. More generally, other methods that infer changes in teammates’ policies could inspire methods to enable GPL to effectively collaborate with adaptive teammates.

Combining different latent variable inference methods. A potential direction to
improve the latent variable inference models proposed in Chapter 5 is to formulate a hybrid latent variable inference method. The design of this hybrid method should be based on the findings from Section 5.8.3 where each inference model is demonstrably more accurate at predicting different subsets of latent variables. The resulting hybrid model should then separately predict each latent variable based on the existing latent variable inference model that delivers the most accurate inference.

**Upside down reinforcement learning for teammate skill level specification.** On top of the teammates generated by BRDiv, upside-down reinforcement learning (Schmidhuber, 2019) (UDRL) provides a promising approach to address the skill level issue in teammate generation for AHT. UDRL is primarily an approach to train policies that must achieve returns according to a specified value. Assuming that a teammate’s skill level can be quantified by their achieved returns, teammates with different skill levels could be generated by adjusting the input returns specified into a policy trained via UDRL. Nevertheless, UDRL methods are currently limited to single-agent RL problems where a trained agent does not need to interact with other agents. For UDRL to help generate teammates, extensions of UDRL to problems where an agent has to interact with others are required.

**GNN-based team configuration generation.** Rather than separately training different collections of policies for each team configuration, a potentially more resource-efficient solution to team generation is learning to assemble individual agents into a team. Much like BRDiv, we can use a centralised critic to encourage different generated team configurations to require different best response policies for effective collaboration. A key component for this approach is to have a critic that estimates the returns of a policy when collaborating with different team configurations of potentially differing sizes. For such a use case, GNNs provide a promising function approximation model. This model should then be combined with methods that learn to form teams of agents under a specific objective (Shu and Tian, 2018).

**Complex environments and human-agent interaction to evaluate BRDiv.** Mirsky et al. (2022) have compiled a list of AHT environments with varying levels of complexity, which all can be utilised to further evaluate BRDiv. From these environments, a potentially exciting possibility is to evaluate against environments where human-agent interaction occurs. For instance, (Lupu et al., 2021) evaluated their method in the Hanabi environment (Bard et al., 2020) where an agent must cooperate with human teammates in a card game. Experiments in these environments are specifically exciting since humans exhibit different behaviours that are often difficult to manually specify.
Appendix A

Teammate Policies For AHT Experiments

A.1 Open AHT Experiments

We implement diverse heuristics to control the teammates in Wolfpack and LBF. Further details of the heuristics used for both environments are provided in the following section. We provide empirical evidence showing that the set of heuristics is diverse and requires significant adaptation to achieve optimal performance by training an agent using the QL baseline against a specific type of teammate and evaluating the resulting policy against different types of teammates. We found that neither policies trained against a specific teammate nor a policy resulting from training against all possible types of teammates could reach the optimal performance against every teammate type.

The results of this experiment for both environments are provided in Figure A.1. As we have done in the team size generalisation experiments, for each approach we periodically save the policies resulting from training and choose the saved policy with the highest performance in training to be evaluated and reported in the heat matrix visualisation. Figure A.1 shows that even for policies trained against all types of agents, none of the resulting policies consistently achieves optimal performance for all teammate types.

On the other hand, for FortAttack we use the 5 pretrained policies provided by Deka and Sycara (2021). In the original work that proposed FortAttack, GNN-based networks were trained to create stochastic policies to control attackers and defenders. Deka and Sycara (2021) subsequently visually analysed the resulting behaviour of the trained policies along different checkpoints and found different adopted by attackers and de-
fencers during the training process. An example policy, which we eventually used as the different agent types for our experiments, was given for each type of strategy adopted by the attackers and defenders.

Figure A.1: **Generalisation performance of a QL agent trained to interact with a single teammate with a fixed type.** We outline the generalisation performance of QL agents trained for a single teammate type in both Wolfpack with a penalty of 0.5 (a) and level-based foraging (b). All experiments here are conducted using 4 seeds. The horizontal axis denotes the type of teammate encountered during training. The “All” versions are trained against a random teammate sampled uniformly over all possible types at the start of each episode. The vertical axis denotes the type of agent used to test the trained policies. Performance in each evaluation environment is scaled between zero and one by performing min-max scaling on each row of the heat matrix.

### A.1.1 Wolfpack

To create a diverse set of teammates for open ad hoc teamwork, we used the following mixture between heuristics proposed by [Barrett et al. (2011)]() for the Wolfpack domain along with RL-based models to control teammates:

- **Random agent (H1):** The random agent chooses its action at any timestep by uniformly sampling the set of possible actions.

- **Greedy agent (H2):** The greedy agent chooses its action following the greedy predator heuristic provided in [Barrett et al. (2011)](). Intuitively, it sets the closest grid cell adjacent to the closest prey from its current location as its destination. It then chooses to move closer to the destination by moving along an axis for which it has the furthest distance from the prey.
A.1. Open AHT Experiments

- **Greedy probabilistic agent (H3):** The greedy probabilistic agent chooses its action following the greedy probabilistic predator heuristic provided in [Barrett et al., 2011]. It chooses its destination in the same way as greedy agents. However, it randomly chooses one of the two available axes to move closer to the nearest prey. An agent’s distance from the prey on each axis is provided as input to a Boltzmann distribution to decide which axis the agent should move along.

- **Teammate aware agents (H4):** This agent follows the teammate aware predator heuristic from [Barrett et al., 2011]. Intuitively, this heuristic assumes all teammates are using the same heuristic. It subsequently computes a hierarchy between agents based on their distance from their targeted prey. The hierarchy determines the sequence in which agents choose their actions. Agents must consider agents’ actions higher up the hierarchy to avoid collisions. An A* planner is subsequently used to compute the action to reach the destination.

- **GNN-Based teammate aware agents (H5):** We train an RFM model with supervised learning to predict the actions a group of teammate-aware agents takes. This was done to avoid the possibly slow running time of the A* planner in teammate-aware agents. During an interaction, it assumes that every agent is a teammate-aware agent and passes their features along with prey locations as input to the network. Agent of this type subsequently uses the representation of its associated node as an input to an MLP which has been trained to imitate the distribution over actions for teammate-aware agents. Our agents subsequently sample the resulting distribution to decide their actions.

- **Graph DQN agents (H6):** We train an RFM-based controller trained by DQN to control a team of agents. It parses the state information following the input preprocessing method for GPL provided in Section 4.2.1 and provides it as input to an RFM. Node representations produced by the RFM are passed into an MLP to compute the action value function of the player associated with the node. Since this type only controls a single agent during an interaction, only the action value associated with the controlled agent is used to take an action.

- **Greedy waiting agents (H7):** This heuristic is similar to greedy agents. However, agents are equipped with a waiting radius sampled randomly between three to five. The greedy heuristic is followed when either the Manhattan distance between the agent and closest prey is more than the waiting radius or when there
is already another teammate inside the waiting radius of the closest prey. Otherwise, the agent will uniformly sample an action until the prey moves away or another teammate comes close to the prey.

- **Greedy probabilistic waiting agents (H8):** Similar to greedy waiting agents, agents of this type are equipped with a waiting radius sampled randomly between three to five. However, the greedy-probabilistic heuristic is being followed when either the Manhattan distance between the agent and the targeted prey is more than the waiting radius or there is already another teammate inside the waiting radius.

- **Greedy team-aware waiting agents (H9):** Similar to greedy waiting agents, agents of this type are equipped with a waiting radius sampled randomly between three to five. However, the teammate-aware heuristic is being followed when either the Manhattan distance between the agent and the targeted prey is more than the waiting radius or there is already another teammate inside the waiting radius.

### A.1.2 Level-based foraging

Like Wolfpack, we create a diverse set of teammate types for level-based foraging, requiring agents to adapt their policies towards their teammates to achieve optimal performance. With level-based foraging, we use a mixture of heuristics (Albrecht and Ramamoorthy, 2013b; Albrecht and Stone, 2019) and controllers trained using the A2C algorithm (Mnih et al., 2016) as our teammate policies. With the heuristic-based agents, their observations are limited to a square patch of grid cells with the agent’s location being the centre of the grid. The size of this observation square is uniformly sampled between $3 \times 3$, $5 \times 5$, or $7 \times 7$. Details of the different types of heuristics used in level-based foraging are provided below:

- **Heuristic H1:** This type of agent follows heuristic $\theta_{j}^{L2}$ proposed by Albrecht and Stone (2019) where agents under this heuristic follow the agent with the highest level if it observes another agent with a higher level than its own. If no agent has a higher level, it follows the farthest observable agent from their locations instead. The controlled agent then computes the object targeted by the leader agent if they follow heuristic H3 provided below and chooses an action that will get closer to the target object. If the agent is already next to the targeted object, it
will choose to pick up the object. If the agent cannot follow the aforementioned rules due to not observing any objects in their observation square, it chooses the leader’s position as its target instead. If no other teammates are observed, it uniformly samples an action from the set of possible actions instead.

• **Heuristic H2:** This type of agent follows heuristic $\theta^F_j$ proposed by Albrecht and Stone (2019) where the controlled agent chooses a leader agent, assumes they follow specific heuristics to choose their targeted object and gets itself closer to the object they think is targeted by the leader. Unlike H1, it chooses an observable agent with the farthest distance from itself as its leader. Furthermore, it assumes that the leader follows heuristic H4 provided below in choosing its target object. Otherwise, the way it chooses its actions when next to the targeted object is the same as in H1. Furthermore, its action selection method when there are no objects or teammate agents in its observation square follows that of H1.

• **Heuristic H3:** This type of agent follows heuristic $\theta^L_j$ proposed by Albrecht and Stone (2019) where the controlled agent targets objects that have the highest level below its level and get closer to the object. If no objects in the set of visible objects are below its level, it chooses to target the item with the highest level instead. Otherwise, it uniformly samples actions from the set of possible actions when no objects are observed in its observation square.

• **Heuristic H4:** This type of agent follows heuristic $\theta^L_j$ proposed by Albrecht and Stone (2019) where the controlled agent targets the farthest visible object from its current location and gets closer to the object. When no objects are visible in its observation square, it samples actions uniformly from the set of possible actions.

• **A2C Agent (H5):** This type of agent is produced by independently training four agents together in level-based foraging using the A2C algorithm (Mnih et al., 2016). Among the four agents, we choose the one with the highest performance compared to others as our A2C-based controller for this type. Also, agents of this type do not have observation squares but receive the whole state of the environment as input to their policy.

• **Heuristic H6:** This type of agent follows one of the heuristics proposed by Albrecht and Ramamoorthy (2013b) for level-based foraging where agents always take actions that take them closer to the closest object in their observation square.
If no objects exist, agents uniformly sample an action from the set of possible actions.

- **Heuristic H7:** This type of agent also follows one of the heuristics proposed by [Albrecht and Ramamoorthy (2013b)](#). In choosing their actions, agents of this type go to the object inside the observation square closest to the centre of all observed players. It follows heuristic H6 when no objects are observed in its observation square.

- **Heuristic H8:** This type of agent follows a heuristic proposed by [Albrecht and Ramamoorthy (2013b)](#) where agents choose the closest object with the same level or lower than their level as their target. If none of such objects exists, the agent uniformly samples an action from its action space.

- **Heuristic H9:** This type of agent follows a heuristic proposed by [Albrecht and Ramamoorthy (2013b)](#) where it scans its surroundings for observable target objects that have at most the same level as the sum of all observable agents’ levels. It then computes the centre of all agents’ locations and chooses a target object with the least distance to the centre of observable agents’ location as its destination. If no possible target exists, it uniformly samples an action from its action space.

### A.1.3 FortAttack

The behaviour of attackers, as well as defenders not under our control, is determined by policies obtained in the experiments from [Deka and Sycara (2021)](#). The policies show distinct behavioural patterns summarised below. More information on how these policies were obtained can be found in [Deka and Sycara (2021)](#).

- **Guard Type 1 - Random guard:** For this agent, we randomly initialise a neural network and used it as the policy network for the guards.

- **Guard Type 2 - Flash laser:** Defenders position themselves in front of the fort and flash their lasers continually. This behaviour is independent of the movement of the attackers.

- **Guard Type 3 - Spread out Flash laser:** Similar to type 1 but instead of positioning themselves in front of the fort the defenders spread out across the whole width of the environment.
A.2 Teammate Generation Experiments

Our teammate generation methods rely on predefined teammate types to measure the learner’s ability to robustly collaborate with previously unseen teammate types. These teammate types differ from each other in terms of their way of selecting which reward-providing coordinates to move towards. The details of these teammate types we use in this experiment is provided below:

- **Guard Type 4 - Smart spreading:** Defenders spread smartly across the defensive zone and only shoot to kill attackers.

- **Attacker Type 1 - Sneak:** Attackers spread out across the whole environment to maximize the likelihood of finding an open space in the defensive line.

- **Attacker Type 2 - Deceive:** Attackers split up their attack. If most of the attackers come from the right, one attacker will try to sneak beyond the defenders from the other side.

- **Guard Type 6 - Ensemble-trained agents:** Guards of this type are trained by letting it interact against attackers uniformly sampled from attacker type 1 and type 2.

- **Heuristic H1.** This heuristic selects the action that gets a teammate closer to the closest reward-providing coordinate.

- **Heuristic H2.** This heuristic selects the action that gets a teammate closer to the furthest reward-providing coordinate from its initial position at the beginning of the episode.

- **Heuristic H3.** A teammate under this heuristic moves towards the closest optimal reward-providing coordinate.

- **Heuristic H4.** H4 moves an agent towards the furthest optimal reward-providing coordinate from a teammate’s initial location.

- **Heuristic H5.** Same as H4, except that the learner considers the suboptimal reward-providing coordinates instead of the optimal ones.

- **Heuristic H6.** Same as H5, except the teammate goes towards the closest suboptimal reward-providing coordinate.
• **Heuristic H7.** At the beginning of the episode, agents under this heuristic randomly select a reward-providing coordinate and move towards it.

• **Heuristic H8.** This heuristic moves a teammate towards the reward-providing coordinate closest to its counterpart agent’s location.

• **Heuristic H9.** Same as H8, but only optimal reward-providing coordinates are considered as the teammate’s destination.

• **Heuristic H10.** This heuristic moves the teammate towards its counterpart agent’s location.

• **Heuristic H11.** This heuristic always randomly selects an action from the teammate’s set of possible actions.
Appendix B

Hyperparameters and Network Sizes

B.1 Fully Observable Open AHT

Details of the neural network architectures used by GPL in Wolfpack and LBF are provided in Figure B.1. Before being processed by the LSTM, the type embedding network passes the input through two fully connected layers. Results from the embedding network are subsequently passed to the GPL component that has the type embedding as input. For the joint action value computation, the singular and pairwise utility computation utilises an architecture provided in Figure B.1b and Figure B.1c. The agent and auxiliary agent models follows the architecture provided in Figure B.1d, Figure B.1e, and Figure B.1f.

To allow a fair comparison between the baselines and GPL, the model architecture used by baselines in Wolfpack and LBF follows the architecture used by GPL. Specifically, baselines pass their input vectors to the architecture in Figure B.1a and subsequently passes the output to an architecture following Figure B.1b to compute the action values. For baselines that use agent and auxiliary agent models, the architecture of these models follows the same architecture provided by Figure B.1d, Figure B.1e, and Figure B.1f.

For MADDPG and DGN, we use networks with similar sizes to those used in open ad hoc teamwork experiments. The only difference is we do not use LSTMs in the network since there is no need for type inference in the MARL approaches. As a result, the architecture used by DGN is simply the architecture used by GNN with the LSTM layers replaced with an MLP with the same output size. The decentralised policy for MADDPG is also similar with the value network architecture of QL with the LSTM replaced by an MLP with the same output size.
Figure B.1: **GPL and baseline architecture details for LBF and Wolfpack.** We provide details of the architecture used in GPLs type embedding network (a), singular utility computation (b), pairwise utility computation (c), edge embedding computation in the agent and auxiliary agent model (d), node embedding computation (e) in the agent and auxiliary agent model, the MLP used by the agent and auxiliary agent model to process the resulting GNN node embeddings (f), the size of the MLPs used for computing the key, query, and value for multihead attention in GNN-QL and GNN-QL-AM (g), and the final layer used in action value computation for GNN-QL & GNN-QL-AM. In all images, **FC** denotes a fully connected layer, **LSTM** denotes an LSTM layer, and the accompanying number denotes the size of the layer. Labels on the arrows indicate the non-linear functions used between the layers while no labels indicate no non-linear functions being applied to the resulting output vectors. With the baselines, we combine the architectures in (a) and (b) to compute the action values while the agent and auxiliary agent model used in some of the baselines follow (d), (e), and (f). With GNN-QL and GNN-QL-AM, the layer defined by (g) and (h) is subsequently used after (a) and (b) to compute the action values.

For training, we use the following hyperparameters for GPL and all proposed baselines:
• $K = 5$ for the low rank factorisation of pairwise utility terms in GPL.

• 8 attention heads were used for DGN, GNN-QL and GNN-QL-AM.

• Data is collected from 16 parallel environments to collect a total experience of 6.4 million environment steps.

• Models are optimised using the Adam optimiser (Kingma and Ba, 2014) with a learning rate of $2.5 \times 10^{-4}$.

• Models are updated every 4 steps on the parallel environment.

• Instead of updating the target networks by periodically copying the joint action value network, target networks are updated using a weighted average of the parameters of the joint action value network. Assuming that $\phi$ is a parameter of the joint action value network, we update the corresponding parameter in the target network $\phi'$ by using $\phi' \leftarrow (1 - \alpha)\phi' + \alpha \phi$, with $\alpha$ set to $10^{-3}$.

• $\epsilon$ for the exploration policy is linearly annealed from 1 to 0.05 in the first 4.8 million environment steps and remains the same afterwards.

• We also use an attention weight regularisation term, $\lambda$, of 0.03 for DGN and a temperature of 0.1 for MADDPG’s Gumbel softmax function.

These hyperparameters and network architectures are obtained by initially searching for a network and hyperparameter configuration which works for QL. After finding an architecture along with a set of hyperparameters which works best for QL, we train a similar-sized architecture with the same hyperparameters for the rest of the baselines. Despite our lack of hyperparameter tuning for the rest of the baselines, we still obtain better performance than QL in almost all cases.

For FortAttack, we initially started the training process with the same hyperparameter setup as LBF and Wolfpack. Due to QL not learning, we decided to increase the size of the network along with running the training process for more timesteps to take into account of the increased complexity of the environment compared to Wolfpack and LBF. Nonetheless, we did not succeed in training QL regardless of the different network sizes and hyperparameters we tried.

We subsequently focused on finding a network architecture and hyperparameters that worked for GPL. A similar-sized network with GPL along with GPL’s training hyperparameters was used for training other baselines. This resulted in the following architecture and hyperparameters for FortAttack:
Appendix B. Hyperparameters and Network Sizes

- $K = 6$ for the low-rank factorisation of pairwise utility terms in GPL.

- 8 attention heads were used for DGN, GNN-QL and GNN-QL-AM.

- Data was collected from 16 parallel environments to collect a total experience of 16 million environment steps.

- Models were optimised using the Adam optimisation algorithm with a learning rate of $1.0 \times 10^{-4}$.

- Models were updated every 4 steps on the parallel environment.

- Instead of updating the target networks by periodically copying the joint action value network, target networks were updated using a weighted average of the parameters of the joint action value network. Assuming that $\phi$ is a parameter of the joint action value network, we updated the corresponding parameter in the target network $\phi'$ by using $\phi' \leftarrow (1 - \alpha)\phi' + \alpha\phi$, with $\alpha$ set to $10^{-3}$.

- The exploration parameter $\epsilon$ was linearly annealed from 1 to 0.05 in the first 8 million environment steps and remains the same afterwards.

- 64 units were used in the fully connected and LSTM layer for LSTM$_{\alpha{Q/Q/p}}$. For comparison, in Wolfpack and LBF we used 100 units.

- 128 units were used for both fully connected layers in MLP$_{\beta}$ and MLP$_{\delta}$. For comparison, we used 70 and 60 units respectively in Wolfpack and LBF.

- 40 and 70 units were used respectively for the first and second layer in the edge and node processing network (GNN$_{\mu/\xi}$) of the agent model. For comparison, we used 30 and 70 units respectively for Wolfpack and LBF.

- We used 128 units in MLP$_{\mu}$ and MLP$_{\nu}$. In comparison, we used 20 units for Wolfpack and LBF.

- Finally, we used 128 units for both fully connected layers involved in the key, query, and value computation under GNN-QL and GNN-QL-AM. For comparison, we used 70 and 60 units respectively for Wolfpack and LBF.

Finally the way these components were assembled into the architectures of the baselines was still the same between FortAttack and the other environments.
B.2 Partially Observable Open AHT

For the architectures that we use in the partially observable experiments, note that we use the same architecture size as in the fully observable experiments for the joint action value and the agent models. For the remaining models in the belief inference model, we use the following architecture:

- For components that are based on GNNs, we use a GNN that has the same size as the GNNs used in agent modelling for the fully observable experiments.

- Components based on multilayer perceptrons have two hidden layers which hidden network size is 100 and 70 respectively.

- The LSTMs used in the belief inference model is similar to the size of LSTMs used for type inference in the fully observable experiments.

Aside from these additional components pertaining the belief inference model, the rest of the hyperparameters used in learning follow the fully observable experiments.

B.3 Teammate Generation

The details of the neural network architecture and method hyperparameters that we use in our teammate generation experiments are provided below:

- We run 16 parallel environments to collect self-play experiences during training.

- 32 parallel environments are used to collect cross-play experiences during training.

- We update the actor and critic for each eight timesteps that elapsed during training.

- $\gamma$ is set to 0.99.

- The action discounting coefficient for TrajeDi is set to 0.85.

- The models of MAA2C are trained using Adam optimiser (Kingma and Ba, 2014) with a learning rate of $10^{-4}$.

- We clip the gradients of the model so that it always lies between -1 and 1.
• Both actor and critic network are implemented as a multilayer perceptron with three hidden layers. The size of the first, second, and third hidden layers are 128, 256, and 128 respectively.

• We also associated different weights to the loss functions that are optimised in BRDiv, TrajeDi, and the baseline that independently trains multiple teammates based on MAA2C. The weights associated to the loss functions for these methods are:
  - For BRDiv, the critic loss function for XP data is set to 1.0.
  - For all methods, the critic loss function for SP data is also set to 1.0.
  - The weight associated to the Jensen Shannon Divergence term for TrajeDi is set to $10^{-4}$.
  - For BRDiv, the weights associated to the term that updates the actor network parameters is set to 25.
  - For TrajeDi and the independent baseline, the weights associated to the loss associated to the actor network is set to 10.
Bibliography


monte carlo. In International Conference on Learning Representations.

in shared multi-agent reinforcement learning. In Beygelzimer, A., Dauphin, Y.,
Liang, P., and Vaughan, J. W., editors, Advances in Neural Information Processing
Systems.

in non-Markovian domains. Citeseer.

wards unifying behavioral and response diversity for open-ended learning in zero-
sum games. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan,
J. W., editors, Advances in Neural Information Processing Systems, volume 34,

agent actor-critic for mixed cooperative-competitive environments. In Advances in
neural information processing systems, pages 6379–6390.


PMLR.

agent variational exploration. In Advances in Neural Information Processing Sys-
tems, pages 7611–7622.

Malik, D., Palaniappan, M., Fisac, J., Hadfield-Menell, D., Russell, S., and Dragan,
A. (2018). An efficient, generalized bellman update for cooperative inverse re-
inforcement learning. In International Conference on Machine Learning, pages
3394–3402. PMLR.

Mirsky, R., Carlucho, I., Rahman, A., Fosong, E., Macke, W., Sridharan, M., Stone,
Conference on Multi-Agent Systems (EUMAS).


