Addressing Deep Reinforcement Learning Empirical Algorithm Performance Evaluations*

Roland Dubb
Supervisor: Associate Professor Jonathan Shock

University of Cape Town
Department of Mathematics and Applied Mathematics

April 2024

*This dissertation is submitted for a Master of Science degree at the University of Cape Town. The work is associated with Shocklab in the Department of Mathematics and Applied Mathematics.

Abstract

Due to the rapidly paced production of deep reinforcement learning (RL) research papers, some recent publications have begun to critique the manner in which RL algorithm performances are evaluated. Building on this recent scrutiny, our work attempts to identify the precise aspects of empirical deep RL algorithm performance evaluations that need attention for improvement. This dissertation begins by briefly introducing the RL problem. Thereafter, we review the literature and discuss recent scrutiny of various aspects of deep RL algorithm performance evaluations. Specifically, we discuss the following aspects: (i) the choice of RL environment, (ii) the measurement of uncertainty, (iii) the collection of data, and (iv) the aggregation of that data. From this discussion we identify two particular problems with RL evaluations, namely the non-linear scaling of algorithm performance scores with the level of skill achieved by that particular algorithm, and the (potentially) biased weighting of scores in the data aggregation process, across RL environments. As multi-agent RL (MARL) presents a recently popular research paradigm whose evaluation procedures have not yet been carefully scrutinised in the literature, we analyse a dataset by Gorsane et al. [1] which documents the evaluation methodologies of many recent deep cooperative MARL publications. This analysis, which reveals several flawed aspects about MARL evaluation, along with the reviewed RL evaluation issues from the literature, motivates for an attempt at constructing an improved RL algorithm empirical performance evaluation guideline.

Multi-criteria decision analysis (MCDA) is discussed as a potential framework that offers a data aggregation procedure that resolves the two aforementioned problems with RL evaluations. Combining the use of MCDA with our insights from the literature, we propose an improved guideline for deep RL empirical algorithm performance evaluations. This is contrasted with another proposed guideline by Gorsane et al. [1] before a proof-of-concept test is conducted. Overall, we aim to move toward the better evaluation of RL algorithms and contribute toward an increased sensitivity to a lack of scientific rigour [2, 3] in the field of machine learning.

Acknowledgements

I would like to begin by expressing my gratitude to Associate Professor Jonathan Shock for supervising this dissertation. I am extremely grateful to him for providing me with the opportunity to work with him. His guidance, support and feedback played a central role in the writing of this work. During the course of the work, Jonathan has also fostered a unique research community of students, academics and members of industry.[1] This has provided all his students with a research community and support network. It has also provided us, and me specifically, with opportunities to engage with the state-of-the-art in the machine learning (ML) research space via internships, research projects, and international collaborations. Jonathan's invaluable efforts and extracurricular insights are a constant source of motivation. I am confident that we will continue to see his research community thrive and grow with years to come.

I also wish to acknowledge the late Emeritus Professor Theodor Stewart. His profound contributions to the field of multi-criteria decision analysis (MCDA) inspired and influenced much of this work. Whilst I only had the opportunity to take one course (Bayesian Decision Modelling) with Prof. Stewart during my honours year, his influence extended to many more of my courses. Prof. Stewart laid the foundations for many of the course syllabi in statistical sciences at UCT. His legacy lives on through the teaching of numerous course note packs that he wrote. I am grateful to have known Prof. Stewart and for the opportunity to have spent time discussing this work with him. I wish to thank him for providing me with guidance and support on MCDA and for demonstrating care for a student's research project. Specifically, I am grateful for him having lent me his MCDA notes and having given me pointers on the use of MCDA in my research area. I extend my condolences to his family and loved ones. I hope that some of Prof. Stewart's legacy endures through this dissertation.

I am also grateful for the help received from one of Professor Stewart's students, Professor Leanne Scott. Prof. Scott assisted in validating my use of MCDA when comparing algorithm performances. She also provided suggestions on why MCDA is useful and robust.

Special thanks goes to Dr Arnu Pretorius for providing me with the opportunity of doing an internship with his research team at Instadeep. Arnu is an especially gifted researcher and motivating manager who leads many research efforts. Having been afforded the opportunity to work with him and his team, I was offered an immense learning opportunity and a window into both industry-level research and the the world of an ML start-up. I am specifically grateful for Arnu's collaborative spirit and dedication to supporting junior-level researchers. I learnt a lot from him and wish him all the best with future endeavours.

I extend my gratitude to my colleagues Rihab Gorsane, Omayma Mahjoub, Ruan de Kock and Siddarth Singh who co-authored the research paper titled *Towards a Standardised Performance Evaluation Protocol for Cooperative MARL* with me during my internship. I am glad to have had the privilege of working with you all. Our achievements during our time together motivated my continued efforts through this dissertation. Specifically, the research paper we wrote together laid the foundations for my continued research in the area of RL algorithm evaluations. I hope to have expanded on that work in this dissertation and offered some further insights into the area. I learnt from each one of you during our time together and am grateful to have experienced your hard-work, dedication and unique skill-sets that shaped the paper to be the success that it was.

Additionally I wish to express gratitude to Matthew Morris for his mentorship during my internship. Matthew's academic rigour and ambition were particularly motivational and I am grateful for having been under his guidance.

This research would not have been possible without the generous funding support provided by UCT and the ETDP SETA. I wish to give thanks to these institutions, without whose funding I could not have completed my masters studies.

[1] See https://shocklab.net/

Lastly, I would like to acknowledge those less fortunate than myself who, due to factors beyond their influence, may not have had the opportunities that I have had to learn, engage and participate in such research.

Declaration of Authorship

About this dissertation titled *Addressing Deep Reinforcement Learning Empirical Algorithm Performance Evaluations*, I, Roland Dubb, confirm that:

1. This work was done wholly while in candidature for an MSc. Applied Mathematics at the University of Cape Town. The work has not been submitted for any other degree or professional qualification at this institution or otherwise.
2. The work contained herein was composed by myself except where explicitly stated otherwise in the text. Where I have consulted the published work of others, this is always clearly attributed. Where I have used the figures, diagrams, or quotes from the work of others, the source is always given.
3. I have acknowledged all sources of help.

Signed: R. Dubb

Signed by candidate

Date: 02/04/2024

Statement of Contribution

This statement provides an overview of the work undertaken in this MSc. dissertation which has resulted in published research. The following contains a summary of the key results and narrative of the published work, as well as a detailed breakdown of my contributions. The dissertation draws on the below work:

'*Towards a Standardised Performance Evaluation Protocol for Cooperative MARL*' [1]

R. Gorsane, O. Mahjoub, R. J. de Kock, R. Dubb, S. Singh, and A. Pretorius, "Towards a Standardised Performance Evaluation Protocol for Cooperative MARL," in Thirty-Sixth Conference on Neural Information Processing Systems, 2022. [Online]. Available: https://openreview.net/forum?id=am86qcwErJm

The above research paper calls for the standardisation of performance evaluation protocols in cooperative Multi-agent Reinforcement Learning (MARL). It examines the evaluation methodologies used in recent deep cooperative MARL research papers and diagnoses, via a meta-analysis of these papers, some concerning trends regarding the level of rigour employed in evaluating algorithm performances in this particular research area. The work ties together insights from the meta-analysis together with surveyed insights from attempts to standardise single-agent reinforcement learning (SARL) algorithm evaluation procedures. It uses these insights to propose a standardised performance evaluation protocol for cooperative MARL.

The paper proposes that such a guideline, if widely used, would improve the validity, replicability and reproducibility of cooperative MARL research. Alongside the research paper and guideline protocol[2] the work includes the publicly released dataset of evaluation methodologies in cooperative MARL[3], an open-source repository containing the tools for evaluating algorithm performances[4] and a reporting template for summarising important information when performing algorithm performance evaluations.

As expressed in the acknowledgements, I would like to reserve special mention to my co-authors on this work: Rihab Gorsane, Omayma Mahjoub, Ruan de Kock, Siddarth Singh and Arnu Pretorius. They have each contributed uniquely with informed knowledge that shaped the paper to be the success that it is, and without their hard-work the paper would not be. As an author, I contributed to initial discussions and planning that lead to the conceptualisation of the paper, the writing of a survey of single-agent RL papers that specifically commented on algorithm performance evaluation methodologies, the data collection of papers for the meta-analysis of cooperative MARL algorithm performance evaluation, some of the cleaning and analysis of those data, the designing of a standardised guideline protocol for cooperative MARL algorithm evaluation by combining insights from the surveyed single-agent RL works and the meta-analysis of MARL works, the critiquing of said guideline and, finally, some of the writing, editing and formatting of the final draft of the paper.

This dissertation extends the work done for the above paper. While the above paper acts as a motivation for this dissertation, the work done here moves beyond the paper and makes new contributions. Only section 2.3, which focuses on MARL evaluation methodologies, follows in a similar fashion to the work seen in Gorsane et al. [1]. That section provides a summary of some of the key findings and plots used to illustrate the state of MARL evaluation. It uses the data provided by Gorsane et al. [1], which documents various MARL evaluation methodologies, to perform this analysis and draw conclusions on the level of rigour of MARL evaluation. This section acts only as a case study of evaluation methodologies conducted in the cooperative multi-agent subset of RL literature. We study it to uncover the level of rigour within this subset in order to motivate improvements to the level of rigour for the broader RL field.

Moving beyond the above paper, this dissertation focuses on the state of evaluation methodologies for the RL field as a whole, rather than just multi-agent RL. Here, we provide a more in-depth background to the problem of evaluation, the RL problem, and the MARL problem as seen in our literature review (section

[2]Found on the project website https://sites.google.com/view/marl-standard-protocol/
[3]https://marl-dataset.notion.site/MARL-dataset-d632230523a74f2793630504ab4542e5
[4]https://github.com/instadeepai/marl-eval

2). The dissertation places more emphasis on systematically reviewing the SARL literature on evaluation, as seen in section 2.2. Resulting from this review, this dissertation forms some differences in opinion from Gorsane et al. [1]. This is especially true with regard to choices made in forming a guideline for better evaluation. Specifically, two problems with RL evaluation are identified, which are not addressed in the above publication. These two problems are the non-linear scaling of algorithm performance scores with the level of skill achieved by that particular algorithm and the potentially biased weighting of scores in the data aggregation process across RL environments.

To address these two problems the dissertation expands on the work done in Gorsane et al. [1] by turning to the framework of multi-criteria decision analysis (MCDA), which is not used in Gorsane et al. [1]. MCDA offers a framework for decision-making that combines the insights from multiple criteria in order to compare a set of alternatives. Hence, this is used to explore the problem of deciding on an algorithm using a set of performance scores on different tasks (the criteria). In this work we demonstrate MCDA to offer a resolution to the aforementioned two problems. MCDA then forms a part of our guideline for improved RL evaluation. We contrast our guideline to the guideline proposed by Gorsane et al. [1] in section 3.3.

Contents

Notation

The notation below is used throughout all sections of this dissertation.

t	Time index
s, s_t, s'	States of the environment
o, o_t	Observations
a, a_t	Actions
π, π'	A policy (sequence of actions, $a_1, a_2, a_3, \dots$)
π^*	The optimal policy
p	Known or unknown fixed probability density
$X, x, x_{k,j}$ and $y(k)$	Performance scores and aggregated score value functions
$\mathcal{S}, \mathcal{R}, \mathcal{A}, \mathcal{O}$	Sets of states, rewards, actions and observations respectively
r	Rewards
$\mathbb{E}, \mathbb{I}, F_X, Q_X$	Expectation, indicator, cumulative distribution and quantile functions
C, J	Cumulative reward and expected cumulative reward
γ	Discount factor hyperparameter (used for rewards)
V, Q or V_π, Q_π	State value function and state-action value function, respectively.
V^*, Q^*	Optimal state and state-action value functions.
θ, ϕ, Ψ, ξ	Parameters (of neural networks)
ϵ	Random noise term
α	Step-size hyperparameter/Learning rate
M	Number of tasks/scenarios an RL algorithm is run within (often during an experiment)
N	Number of runs an RL algorithm is run for on a specific task/scenario
W	Number of agents in some multi-agent system
$\mathcal{B}$	Experience Replay Buffer
L	Some loss function (objective function)
i, j, k	Indices of agents in a system, tasks in a set of tasks, and algorithms in a set of algorithms
E	Number of algorithm evaluation episodes per training timestep
$\Theta, \mathcal{L}$	Set of tasks and set of algorithms respectively
ω, q	Weights used in value functions
g	Normalisation function
σ	Standard deviation

1 Introduction

Deep Reinforcement learning has emerged in recent years as a field of study that can address large scale decision-making problems [4, 5, 6]. In particular, the reinforcement learning (RL) problem refers to how a decision-making agent can optimally select a sequence of actions in an uncertain environment so as to maximise a numerical reward signal [7]. While classical RL methods were applied to small discrete state-spaces [7], *deep* RL has made use of function approximators such as deep neural networks to scale up the classical methods. The use of these function approximators within RL allows for the processing of higher dimensional state-action spaces and has brought about many contemporary deep RL algorithms [8, 9, 10, 11]. The recent widespread scale of publicity and popularity of research into the deep RL problem frames the basis of, and motivates for, the study in this dissertation.

In the context of the RL field, recent scrutiny has been leveraged against the level of rigour and methodological standards that the RL community has upheld [1, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]. In particular, the problem of empirically evaluating the performances of deep RL algorithms has come under criticism. Past work has sought to address aspects of this problem. In this work we seek to draw on those insights and extend them to move toward the construction of a guideline which standardises deep RL empirical algorithm performance evaluations.

The field of RL research falls into a broader research context, that of machine learning (ML). As with RL, the quantity of ML research has seen massive growth in recent years. This growth has been met with calls for increased rigour. Sculley et al. [2] suggest that the rate of empirical advancement in the field of ML may not have been matched by a consistent increase in the level of empirical rigour. This lends itself to the view that the rate of progress in the ML field may have been distorted. For instance, '*wins*', where an algorithm demonstrates an improved performance over another, have been emphasised as a metric of success [21, 24]. This contrasts with the more scientific pursuit of knowledge, such as answering *how and why* it is that an algorithm should perform better than another. This problem presents itself in the specific case of RL.

In this work we set out to define the RL problem and extend this to define the multi-agent RL (MARL) problem. With these in mind we study the empirical algorithm performance evaluations of both contexts. Our literature review surveys what work has been done to improve rigour in the area of single-agent RL (SARL) evaluations. As MARL presents a recent subset of RL research which is lacking similar studies (on the problem of evaluation), we use this subset for a case study into the level of rigour of recent RL publications. To this end we study a dataset [1] documenting the evaluation procedures of deep cooperative MARL empirical algorithm performance evaluations.[5] As we show the problem of evaluating deep *cooperative* MARL algorithms to be equivalent to that of evaluating SARL algorithms, the conclusions drawn from this analysis hold relevance for the broader field of RL evaluation. Hence, we combine these insights with those from our SARL evaluation literature review and attempt to construct a guideline for improved deep RL empirical algorithm performance evaluations.

In order to construct this guideline, we make use of methods found in the practise of *multi-criteria decision analysis* (MCDA). MCDA is a framework for resolving decision problems that have multiple criteria for evaluation [25, 26]. As we attempt to resolve the decision problem of selecting an algorithm from a set of algorithms using the criteria provided by the algorithm performances on a set of tasks, we show that MCDA offers an appropriate framework for enhancing RL algorithm evaluations.

We hence demonstrate that the use of MCDA provides a means of coherently measuring an algorithm's performance across a range of RL tasks and a consistent means of comparison between the measurements of the different algorithms' performances. More specifically, we demonstrate that the MCDA framework offers a coherent way of aggregating algorithm performances across tasks that takes into account different reward scales across tasks as well as the non-linear scaling of performance scores with task difficulty.

[5] The data is collected from a wide selection of recent MARL research papers and was collected in Gorsane et al. [1].

Overall, we aim to understand and answer the following questions:

i Are there specific problems with deep RL empirical algorithm performance evaluations, and why are these an issue?

ii What is the level of rigour in deep cooperative MARL empirical algorithm performance evaluations?

iii Can MCDA be used to benefit a guideline for coherent empirical algorithm performance evaluations, and if so, how?

iv Given the aforementioned guideline, can this be used to effectively evaluate deep RL algorithms, and how?

We endeavour to answer the first of these questions via our literature review into RL evaluation. We use the literature review to break down the components that go into a deep RL algorithm performance evaluation. This will allow us to systematically study the performance evaluations to discover our hypothesised specific areas of issue. The issue of providing a *normalisation* function like $\frac{x-\min_k x_k}{\max_k x_k - \min_k x_k}$ [27] for algorithm k's performance score x_k presents a specific area that we hypothesise to be problematic [21]. We hypothesise that the literature and data gathered will demonstrate a poor level of rigour in the empirical evaluation of deep cooperative MARL algorithm performances [1]. The dissertation will henceforth use this as motivation to demonstrate how MCDA can be used to design the aforementioned guideline. Here, we hypothesise an effective use of this coherent framework for the benefit of aggregation across tasks with many different reward functions. We will discuss the benefits that MCDA provides to RL evaluation, contrast how our guideline differs from one provided by Gorsane et al. [1] and discuss the advantages and shortcomings of both of these. Lastly, we conduct a proof-of-concept test to study our guideline's hypothesised efficacy. Here, we employ our guideline to evaluate the empirical performances of three deep RL algorithms on three RL tasks. We then use our guideline to compare the algorithms' performances. Answering these questions will lead to answering our overall aim of addressing the problem of empirical algorithm performance evaluation in deep RL.

1.1 Overview and Scope

The dissertation begins with a literature review, with section 2.1 introducing the RL problem and multi-agent RL problem. This leads onto section 2.2 providing a thorough survey of the literature on single-agent RL evaluation. This section aims to address question (i) above and uncover both the components of RL algorithm evaluation and any problems with RL algorithm evaluation.

Section 2.3 studies the dataset found in Gorsane et al. [1]. This section provides commentary on the evaluation methods (and shortcomings) of the multi-agent literature. The study here aims at answering our second research question on the level of rigour of deep cooperative MARL algorithm performance evaluations.

It is important to mention that this dissertation focuses on *empirical performance* evaluations of deep RL algorithms. In studying *empirical performance*, we unfortunately eschew more theoretical performance studies of properties of RL algorithms. A limited discussion of these can be found in section 2.1 for MARL algorithms in particular. Furthermore, we limit our focus to the *deep* RL field due to its current popularity and its capabilities beyond classical, tabular RL solution methods. For our research on MARL, we limit our focus to only *cooperative* MARL. This is both due to its current popularity and its problem equivalence with the performance evaluations of SARL algorithms. This is discussed in sections 2.1 and 2.3.

Section 2.2 outlines that performance evaluations in the RL field have several complications with regard to the aggregation of performance data across tasks. Hence, a possible solution to this problem of aggregation

is reviewed in section 2.4. Here, the field of multi-criteria decision analysis (MCDA) is introduced as a potential solution method for resolving the aggregation problem.

Taking our components from the literature review, section 3.1 compiles a guideline for performance evaluations of deep RL algorithms. This makes use of MCDA and the advice found in the literature review. This guideline can be found in section 3.2 and is discussed in section 3.3. This section compares the use of our guideline to that found in Gorsane et al. [1] and seeks to respond to our third research question of whether MCDA can be used to benefit a guideline for better empirical performance evaluations in the field of deep RL.

Lastly, section 4.1 outlines our method for how to make use of our guideline and subjects it to RL algorithm performance data. The results of this test are reviewed in section 4.2 and discussed in section 4.3. This proof-of-concept test works to answer our fourth and final research question on whether our MCDA-enhanced guideline can be used effectively with deep RL algorithm performance data. Finally, section 5 offers some concluding remarks in response to the findings discussed in sections 3.3 and 4.3.

2 Literature Review

Addressing the problem of evaluation in deep RL first requires that we define the RL problem to the reader. Here, we seek to provide context to the problem of RL evaluation via the great expanse of recent deep RL literature which motivates this dissertation. This section begins by briefly introducing the RL problem. We also briefly introduce the multi-agent RL problem as a subset of the RL research field which has garnered much recent attention due to its capacity to solve RL problems at scale [4].

Having established the context of our problem, we then focus on the primary issue of this dissertation: deep RL empirical algorithm performance evaluations. Subsection 2.2 defines the problem of evaluation in RL. This is accompanied by the review of works that address the lack of rigour and methodological standards in RL evaluations. We look to these works to systematically understand the components of RL evaluation and answer our first research question on whether there are specific problems we can identify with evaluation that still need addressing.

Subsequently, we will look at the cooperative MARL literature and specifically study a dataset by Gorsane et al. [1] which documents the evaluation procedures of much current MARL literature. From this dataset, we will diagnose what problems and inconsistencies may arise in cooperative MARL algorithm performance evaluations. This serves as a case study of a relatively recent subset of deep RL literature whose evaluation procedures have not yet specifically been analysed. The analysis of this dataset will address our second research question about uncovering the level of rigour in deep cooperative MARL empirical algorithm performance evaluations.

Having defined and diagnosed our problem, we will review the mathematical mechanisms of multi-criteria decision analysis (MCDA) in subsection 2.4. MCDA will later be used in addressing our problem and providing a coherent guideline for improved algorithm evaluations. This will be demonstrated in section 3.

2.1 An Introduction to Reinforcement Learning

The RL problem forms the basis of our interest in this dissertation. We look to review the RL problem and how its solution algorithms are evaluated with the view to making improvements to this process. Before examining MARL we study the single-agent RL problem.[6] Here, we provide a brief introduction to the RL problem which will provide context to the problem of RL algorithm evaluation that attracts this dissertation's specific interest.

2.1.1 Single Agent Reinforcement Learning

Some of the successes of single-agent RL (SARL) have lead to an increased interest in the field in recent years. This is especially true since the adoption of *deep* SARL which makes use of powerful neural networks to scale up the capacity of the previously tabular algorithms [8]. This increased interest acts to motivate the study of RL for this dissertation.

To name a few successes, an RL algorithm has beaten the world champion at the game of Go [28], RL algorithms have seen success in the field of robotics [29] and success at the Atari games [8]. Here, we provide a brief introduction to SARL.[7] This introduction forms the basis of the principles and algorithms which have lead to the above successes.

[6] We will later see that cooperative MARL extends the capabilities of SARL to help solve problems of a larger scale by using features such as reward and parameter sharing between agents [4, 5, 6]. The realisation that the agents, in the cooperative setting, attempt to achieve some common objective means that lessons from single-agent algorithm evaluation literature can be extended to the cooperative multi-agent case.

[7] A more detailed overview can be attained via Sutton and Barto [7] and via the lecture series by Silver [30].

2.1.1.1 The Reinforcement Learning Problem

The RL problem refers to the problem of how a decision-making agent in an uncertain environment can optimally select a sequence of actions so as to maximise a numerical reward signal [7]. The maximisation of cumulative reward provides the central tenet of RL. Mathematically, SARL is often modelled as a Markov decision process (MDP) which can be defined as the tuple of (i) states of some environment, $s \in \mathcal{S}$, (ii) actions from some action set unique to a state, $a \in \mathcal{A}$, (iii) a reward function, $r(s_t, a_t, s_{t+1}) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow \mathbb{R}$ which is bounded by some $[r_{min}, r_{max}]$, (iv) a transition probability function between states (given actions and previous states) $p(s_t|s_{t-1}, a) : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{P}(\mathcal{S})$ and (v) a discount factor, $\gamma \in [0, 1]$ [31]. This gives the tuple $(\mathcal{S}, \mathcal{A}, r, p, \gamma)$ [32].

Figure 1 illustrates how the RL agent learns via trial-and-error learning whilst engaging with a numerical reward signal that it uses to *reinforce* the actions that it should take to accumulate the greatest amount of cumulative reward. The agent, which finds itself in an uncertain changing environment and some state s, engages in trial-and-error learning by sampling from its action set $\mathcal{A}$. This will, in-turn, impact its environment. Its state may therefore change, leaving it in a new state with some amount of reward as a consequence of its sampled actions. This feedback loop (via the reward signal from the environment) assists the agent to improve its action selection policy and, therefore, *learn*.

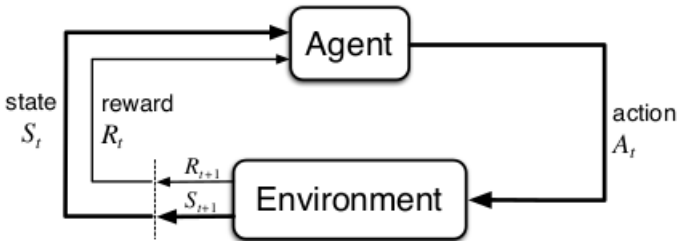

Figure 1: The agent-environment loop, as presented in Sutton and Barto [7]. This illustrates the process of trial-and-error learning that the SARL agent engages in as well as the reward signal which assists the agent to reinforce which actions to take to accumulate the greatest amount of cumulative reward.

Hence, the goal of a RL agent is to model some policy function, $\pi(a|s) : \mathcal{S} \longrightarrow \mathcal{A}$, that optimally solves the given task (so as to maximally accumulate reward) [7, 33, 34]. The policy function decides how an agent will select actions from only the knowledge of its state of the environment. One can further define a state value function $V_\pi(s)$ associated with a given policy π. This is defined as the expected cumulative reward for a given policy, π, when starting from state s and continuing to select actions according to π. The following gives the discrete-time *state value function* for any state s_t and policy π as [7]:[8]

$$\begin{aligned} V_\pi(s_t) &= \mathbb{E}_\pi\left[C_t|s_t\right] = \mathbb{E}_\pi\left[\sum_{f=0}^{\infty} \gamma^f r_{t+f+1}|s_t\right], \forall s_t \in \mathcal{S} \\ &= \sum_{a_t} \pi\left(a_t \mid s_t\right) \sum_{s_{t+1}, r_{t+1}} p\left(s_{t+1}, r_{t+1} \mid s_t, a_t\right)\left[r_{t+1} + \gamma V_\pi\left(s_{t+1}\right)\right] \end{aligned} \tag{1}$$

Similarly to the state value function, there exists a related quantity, the state-action value function $Q_\pi(s, a)$ [7].[9] This is defined to be the expected cumulative reward obtained by executing the action a from state s and following the policy π thereafter. The approximation of these two value functions forms the basis of many RL algorithms, which are known as *value-based methods*. The discrete-time *state-action value function* is [7]:

[8] This recursive expression is known as the Bellman equation for $V_\pi(s)$ [7].
[9] This is sometimes referred to as the action value function only.

$$
\begin{aligned}
Q_\pi(s_t, a_t) &= \mathbb{E}_\pi \left[C_t | s_t, a_t \right] = \mathbb{E}_\pi \left[\sum_{f=0}^{\infty} \gamma^f r_{t+f+1} | s_t, a_t \right] \\
&= \sum_{s_{t+1}, r_{t+1}} p\left(s_{t+1}, r_{t+1} \mid s_t, a_t\right) \left(r_{t+1} + \gamma V_\pi\left(s_{t+1}\right)\right)
\end{aligned} \tag{2}
$$

To obtain the optimal policy in RL one makes use of the policy improvement theorem [7]. This suggests that if π and π' are any two policies with corresponding value functions having $Q_\pi\left(s, \pi'(s)\right) \geq V_\pi(s), \forall s \in \mathcal{S}$ then, the policy π' must be better than or equivalent to π. The optimal policy π^* directly corresponds to the optimal value function Q^* [7]. Defining the optimal value function makes use of the recursive Bellman optimality equations. These assist in recursively expressing the value of a state in terms of the value of its successor states. This leads to defining discrete-time Q^*, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ as [7]:

$$
\begin{aligned}
Q^*(s_t, a_t) &= \max_\pi Q_\pi(s, a) \\
&= \mathbb{E}\left[r_{t+1} + \gamma V^*\left(s_{t+1}\right) | s_t, a_t \right] \\
&= \mathbb{E}\left[r_{t+1} + \gamma \max_{a_{t+1}} Q^*\left(s_{t+1}, a_{t+1}\right) \middle| s_t, a_t \right] \\
&= \sum_{s_{t+1}, r_{t+1}} p\left(s_{t+1}, r_{t+1} \mid s_t, a_t\right) \left[r_{t+1} + \gamma \max_{a_{t+1}} Q^*\left(s_{t+1}, a_{t+1}\right) \right]
\end{aligned} \tag{3}
$$

As seen in Figure 2, the Bellman optimality equation for Q^* averages over the different possibilities of states and takes the maximum over possible next actions depending on the estimated value of the subsequent state-action pair, estimated using Q^*. One can imagine that providing these estimates for Q^* for each state and action pair means that the tractability of such a problem, given a large state-action space, is quite complex.

Hence classical SARL, which made use of tabular solution methods, was often limited in application to small, discrete state-action spaces [7]. Contemporary SARL algorithms [8, 9, 10, 11], have therefore addressed this scalability problem through the use of powerful function approximators such as deep neural networks to approximate agents' value functions or policies. This type of supervised learning approach requires large amounts of data, which is often collected through an agent's experiences of various state-action-reward trajectories. It is the improvements to function approximators like deep neural networks that have lead to the recent successes of RL, mentioned briefly above. This is what motivates our study into RL.

2.1.1.2 Solving the RL Problem

One important SARL solution method that has been adapted to make use of neural networks is *Q-learning*. This refers to the estimation of state-action value functions. Q-learning makes use of the recursive nature of the Bellman equations to provide an iterative update equation for state-action value functions this provides the following bootstrapped update expression (where α is some step-size hyperparameter) [7]:

$$
Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha \left[r_{t+1} + \gamma \max_a Q\left(s_{t+1}, a\right) - Q\left(s_t, a_t\right) \right] \tag{4}
$$

The update expression above means that an agent can update its model for the expected cumulative reward for a given state s and action a, $Q(s, a)$. All it needs is some model for the function Q and its current experienced reward for taking action a from state s. The function Q can be estimated using a deep neural network as seen with the *Deep Q Networks* (DQN) solution method [8, 35]. As an agents explores its environment it collects its trajectories of observations, actions and rewards and stores these in a memory

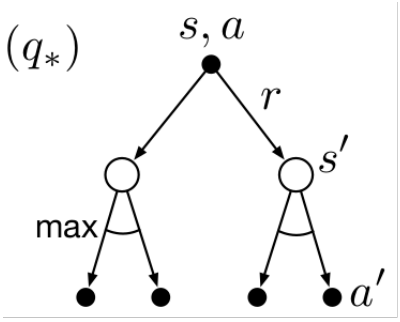

Figure 2: This figure, from Sutton and Barto [7], represents the approximation of the optimal state-action value function, Q^*, for some policy π, for an example discrete MDP. Given that the figure represents the flow through time for an agent from top to bottom, the agent may start in some state s and take some action a (corresponding to the root node). The estimation of Q^* then, according to the Bellman optimality equations, takes the expectation over the rewards from the agent transitioning to any of the next possible states (which are represented by the white dots). This is added to the expected optimal cumulative reward from the next state s' (once discounted by a product of γ). This is estimated by taking the maximum of a current Q^* value estimate for the agent's next action a' from s' (seen on the third row of Equation 3). This provides the bootstrapped, recursive expression for Equation 3.

buffer. For instance tuples of states, actions, rewards and subsequent states like (s_t, a_t, r_t, s_{t+1}) are collected and stored in a memory buffer of fixed size. The collection of these tuples are randomly sampled from, to provide training data to update the parameterised Q_θ function. As the Buffer storing the tuples is of fixed size, newly experienced, more recent tuples are added to the end of the Buffer at the expense of removing the oldest tuples of agent experience. This allows the agent to train (and improve its Q function) on data more relevant to its current policy.

After successfully estimating a Q function using the data from the agent's experience, the agent need only select actions such that they are *greedy* with respect to Q. This yields the policy below [36]:[10]

$$\pi_\theta\left(a_t|s_t\right) = \delta\left(a_t - \underset{a_t \in \mathcal{A}(s_t)}{\operatorname{argmax}}\, Q_{\pi_\theta}\left(s_t, a_t\right)\right) = \begin{cases} 1 & \text{if } a_t = \underset{a_t \in \mathcal{A}(s_t)}{\operatorname{argmax}}\, Q_{\pi_\theta}\left(s_t, a_t\right) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

In solving the SARL problem the process of iterating between estimating value functions and then selecting the corresponding greedy policies is known as *generalised policy iteration* [7]. The aforementioned policy improvement theorem used in conjunction with the generalised policy iteration process[11] will lead to convergence to the optimal policy and corresponding optimal value function.

The above Q-learning solution method is known as a value-based method in the context of a taxonomy of SARL solution methods. The next class of solution methods that we will discuss are policy-gradient methods which, in contrast to value-based methods, primarily estimate the parameterised policy function of the agent $\pi_\theta(a_t|s_t)$ [36]. Both these types of methods fall into what is known as the model-free solution class of solution methods to the SARL problem [7]. As with the Q function, the policy function is often approximated using a neural network with parameters, say $\theta \in \mathbb{R}^d$. The policy can then be estimated by performing stochastic gradient ascent on the RL agent's expected cumulative reward function, $J(\theta)$. This yields [7]:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J\left(\theta\right) \tag{6}$$

[10] Here, δ represents a Dirac delta function.
[11] and some exploration method

Here $J(\theta)$ can be estimated using the agent's collected (Monte-Carlo) samples of cumulative rewards under some policy $\pi_\theta(a|s)$ [7]. Hence for an episode of sampling states, actions and rewards, using policy π, up till time T, we have objective function [7, 36]:

$$J(\theta) = \mathbb{E}_{\pi_\theta(a_t|s_t)} \left[\sum_{t=0}^{T-1} r\left(s_t, a_t, s_{t+1}\right) \right] \tag{7}$$

To calculate the gradient $\nabla_\theta J(\theta)$ one need make use of the *policy gradient theorem* as the above expression is non-differentiable [9]. This theorem states that the gradient of the expected cumulative reward objective function $J(\theta)$ (with respect to θ) is equal to the average gradient of the policy multiplied by the return received [7, 36]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta(a_t|s_t)} \left[r_t\left(s_t, a_t, s_{t+1}\right) \nabla_\theta \ln \pi_\theta\left(a_t|s_t\right) \right] \tag{8}$$

The gradient can be estimated using the Monte-Carlo sampled trajectories of states, actions and rewards collected by the agent. The above method is known as *REINFORCE* [9].

Thus, we have discussed two different, contemporary types of solution method to the SARL problem, DQN and REINFORCE, both of which make use of deep neural networks to solve the problem of scaling to large state-action spaces. A third model-free class of solution method combines the policy-gradient method and value-based method. This replaces the Monte Carlo estimate of cumulative reward in the above expression, seen in Equation 8, with a *critic* (the estimated Q value function) such that [7, 9, 36]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta(a_t|s_t)} \left[Q_{\pi_\theta}\left(s_t, a_t\right) \nabla_\theta \ln \pi_\theta\left(a_t|s_t\right) \right] \tag{9}$$

This is known as an *actor-critic* method [9] and has the advantage of reduced model variance as compared to more vanilla policy-gradient methods. Furthermore, in *advantage actor-critic* methods [10], one can subtract a baseline from the sampled cumulative reward. This yields the advantage function A_t which further reduces the variance in the updates. The policy parameters are updated in accordance with how much better or worse the sampled policy performs (measured by r) as compared to the critic (measured by the state value function V). This yields:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta(a_t|s_t)} \left[\underbrace{\left(r_t\left(s_t, a_t, s_{t+1}\right) - V_{\pi_\theta}\left(s_t\right)\right)}_{A_t} \nabla_\theta \ln \pi_\theta\left(a_t|s_t\right) \right] \tag{10}$$

In addition to DQN [8, 35], REINFORCE [9] and advantage actor-critic methods [9, 10], some other contemporary model-free SARL solution algorithms include *proximal policy optimisation* (PPO) [37], *Deep Deterministic Policy Gradients* (DDPG) [38] and *Soft Actor-Critics* (SAC) [39]. Each of these provides an example of an algorithm which one may seek to evaluate the performance of when selecting the appropriate algorithm for a given task.

2.1.1.3 Types of RL Solution Algorithms

In addition to model-free algorithms there exists the *model-based* RL solution algorithm paradigm. Some algorithms hybridise model-free and model-based solution methods. These include the Dyna [11] and control as hybridised inference algorithms [40]. Other taxonomic categories in RL solution algorithms include offline vs online, value-based vs policy-based, Monte Carlo learning vs Temporal Difference learning, fully observable vs partially observable, off-policy vs on-policy, single agent vs multi-agent [7] and iterative vs amortised inference [32]. We briefly elaborate on some of these taxonomic categories here.

- Value-Based vs Policy-Based: As discussed, DQN [8, 35] provides an example of a value-based method which learns a Q state-action value-function, from which an ϵ-greedy policy can be used. In contrast, policy-based algorithms like REINFORCE [9] primarily model policy function $\pi_\theta(a_t|s_t)$ which directly ascribe actions given states.

- Model-Free vs Model-Based: As seen with the above DQN [8, 35] and REINFORCE [9] algorithms, model-free RL algorithms are able to learn an agent's policy or value-function without explicitly constructing, and without explicitly learning, a model of the environment's dynamics. In other words, the value functions or policies are learnt using the agent's experienced sample trajectories. This is useful in contexts where it is easy and cheap for an agent to gain experience of its environment [34]. In contrast, model-based RL algorithms form an explicit model of the state dynamics of its environment [7].[12] This may offer a more sample-efficient approach since the agent may use its model to simulate trajectories of states, actions and rewards to estimate expectations and improve its policy (model-based planning).

- Online vs Offline RL: Whilst online RL algorithms train on the sampled experiences of the agent in its environment, offline algorithms are trained on static and previously collected datasets of transition tuples (s_t, a_t, r_t, s_{t+1}) from some policy which is potentially unknown [41].

- Monte Carlo Learning vs Temporal Difference Learning: Monte Carlo (MC) methods use the agent's entire sampled trajectories to iteratively update the value function estimates based on observed returns for entire episodes [7]. They require the RL problem to have a terminal state so as to obtain the episode return. An example of an update rule for a Monte Carlo method with learning rate α and with cumulative episode return at terminal state T, C_T is: $Q(s,a) \leftarrow Q(s,a) + \alpha\left[C_T - Q(s,a)\right]$. In contrast, temporal difference (TD) methods use a bootstrapped estimate from the Bellman equations to resolve the need for terminal states. To estimate the value of a state, TD learning uses the current-time estimate of the value function for its expected successor state (bootstrapping). This therefore requires only experience tuples of one timestep to learn, for instance (s_t, a_t, r_t, s_{t+1}). The Q-learning update, seen in Equation 4 demonstrates 1-step TD learning.

- Fully Observable vs Partially Observable: This differentiates between algorithms that have access to the full state of the environment s or only partial access via some observation o which is dependent on the state.

- Off-Policy vs On-Policy: Off-policy algorithms are trained on data from a policy other than the one that is trying to be improved for execution. In contrast, on-policy algorithms learn from the current policy executing in the environment. DQN [8, 35] is an example of an off-policy algorithm while PPO [37] is an on-policy algorithm.

Lastly, some RL algorithms are single-agent and some are multi-agent. Next we will provide a brief background to the multi-agent RL problem before discussing the problem of evaluating the performances of RL algorithms, which forms the main focus of this dissertation.

[12] A model of state dynamics may be learnt alongside a reward model, although the reward model is often not needed.

2.1.2 Multi-Agent Reinforcement Learning

We now extend SARL to the multi-agent case. In this section we will briefly introduce the MARL problem. This work focuses on the cooperative setting whereby agents cooperate to jointly achieve some common objective. This can afford MARL the possibility to perform tasks of a larger scale than SARL [4, 5, 6].

Contemporary MARL research is often found in the video game sphere. For instance, MARL has achieved grand-master level performance at the StarCraft II strategy game with the *AlphaStar* algorithm [42]. MARL has found many other application contexts, for instance in traffic light management using networked system control [43], multi-objective contexts like common-pool resource management in which agents may compete with one another for a shared resource [44] and large centralised control problems like for electricity, communications and transportation [43, 44, 45, 46]. More generally, MARL may offer performance benefits in (i) large decentralised control problems, where communication may cause a delay [43, 47], (ii) it may offer benefits in scenarios where there are many competing interests [44, 47] and (iii) it offers benefit in large centralised control tasks in which the state-action space is intractable for SARL [47]. For these reasons MARL offers an interesting paradigm for large system control. These examples act to motivate for the development of better MARL algorithms, for understanding the current state of the field and for improving the rigour of algorithm evaluations.

MARL's history finds its footholds in game theory [47]. Originally this sought to analyse games of multiple agents competing against one another (competitive games). However game theory has evolved into a framework for analysing interactions between multiple agents in general. Different types of games can be classified into three broad categories: competitive, cooperative and mixed.

A theoretical understanding of MARL often begins with *Markov games* [48]. These provide an extension of the MDP to the multi-agent setting. A Markov game is defined by the tuple $\left(W, \mathcal{S}, \left\{\mathcal{A}^i, r^i\right\}_{i=1}^n, p, \gamma\right)$. Here, W gives the number of agents which are indexed by i, $\mathcal{S}$ gives the state-space of the task observed by all agents, $\mathcal{A}^i$ provides the action space for agent i such that the joint action space is the cross product $\mathcal{A} = \otimes_{i=1}^{W} \mathcal{A}^i$, $r^i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow \mathbb{R}$ gives agent i's reward function, $p(s_t \mid s_{t-1}, \mathbf{a}_{t-1}) : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{P}(\mathcal{S})$ provides the transition probability function between states and finally, $\gamma \in [0, 1]$ provides a discount factor [31]. Note in Figure 3 that the environment is affected by the joint actions of all the agents. This provides an additional complication for learning policies for agents in multi-agent contexts.

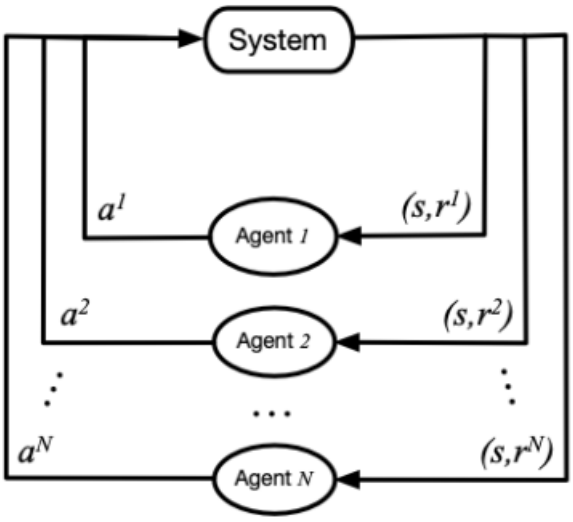

Figure 3: The agent-environment loop for a Markov Game, as presented in Zhang et al. [49]. This illustrates the process of trial-and-error learning that the MARL agents engage in as well as the reward signal which completes the feedback loop with the environment.

It is often common for agents to not have a global view of the state of their environment. Hence MARL

is often defined in terms of a *partially observable Markov game* [31]. This is defined by augmenting the Markov game tuple with each agent's observation space $\mathcal{O}_i$ which has transition probability defined by the probability function $p_o = p_o(\mathbf{o}_t|\mathbf{a}_{t-1}, s_{t-1})$ [50]. This gives the tuple $\left(W, \mathcal{S}, \left\{\mathcal{A}^i, r^i, \mathcal{O}^i\right\}_{i=1}^n, p, p_o, \gamma\right)$.

The MARL game categories, mentioned above are defined by the reward structure assigned to each of the multiple agents. The reward functions differ between the cooperative, competitive and mixed settings [31]. The competitive setting often has agent rewards sum to zero such that $r = \sum_{i=1}^{W} r^i(s_t, a_t, s_{t+1}) = 0$. The mixed setting, which is often referred to as a *general-sum* game has no general way of defining its reward structure as it can encompass any chosen structure. Lastly, the cooperative setting often has that all agents receive the same reward $r = r^1 = ... = r^W$.[13]

In this dissertation we limit our focus to cooperative MARL. This is because cooperative MARL provides the focus of the vast majority of contemporary MARL research. Cooperative MARL also provides a problem equivalence to SARL in the context of algorithm performance evaluations. This problem equivalence means that the lessons learnt from a cooperative MARL algorithm evaluation literature review will be more broadly applicable to RL evaluation in general.

Next, we briefly introduce cooperative MARL. This subsection will provide context into the subset of RL algorithms that we will later investigate when we study the dataset by Gorsane et al. [1] on evaluation methods in the cooperative MARL research field. Our analysis of that dataset aims to lend itself to an improved understanding of the evaluation methodologies used in contemporary RL research.

2.1.2.1 The Cooperative Multi-Agent Reinforcement Learning Problem

Similarly to the single-agent case, the cooperative MARL problem is one of how a number, W, of decision-making agents in an uncertain environment can each select an action sequence so as to together maximise the total amount of reward accumulated. More formally, this environment is defined as a Decentralised Partially Observable Markov Decision Process, a *Dec-POMDP* [49, 50, 51, 52]. A Dec-POMDP is defined similarly to the partially observable Markov game except that the reward function is not unique for each agent and there is instead a single joint reward function, $r(s_t, \mathbf{a}_t, s_{t+1}) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow \mathbb{R}$. A Dec-POMDP is therefore defined by the following tuple, a modification of the partially observable Markov game, $\left(W, \mathcal{S}, \left\{\mathcal{A}^i, \mathcal{O}^i\right\}_{i=1}^n, r, p, p_o, \gamma\right)$ [51].

In this framework, at each timestep each of the W agents, indexed by $i \in \{1, ..., W\}$, receives a local observation o_t^i and takes some action a_t^i. As seen in Figure 3, the joint action of all agents, $\mathcal{A}$ affects the state of the environment which may cause it to change and emit some amount of reward r_t, which is received by all the agents [47]. As with SARL, the central tenet in cooperative MARL is for each of the agents to find an optimal policy, $\pi^i : \mathcal{O}^i \longrightarrow \mathcal{A}^i$, so as to maximise their cumulative reward [50].

2.1.2.2 Solving the Cooperative MARL Problem

Solving the cooperative MARL problem involves each agent selecting a policy that attempts to maximise its rewards. However, in attempting to maximise each agent's reward, an algorithm may find a sub-optimal solution for the collective. A *best response* is therefore defined in game theory [31]. This is where the policy of an agent is as good as it can be where all other agents' policies remain the same. If we denote the joint policy of all agents as $\pi = \{\pi^1, ..., \pi^W\}$ and the policy of all agents apart from agent i as π^{-i} then, the best response is where agent i follows its optimal policy π^{i*} with respect to the joint policy π^{-i} of all the other agents such that for all state $s \in \mathcal{S}$ and policies $\pi^i \in \Pi^i$ [31]:

[13] An average or sum of individual agent rewards is also possible.

$$V_{\pi^{i*},\pi^{-i}}(s) \geq V_{\pi^{i},\pi^{-i}}(s) \tag{11}$$

Here V denotes the value function which is defined similarly to SARL except for its dependence of the joint actions of all agents, π [31]:

$$V^{i}_{\pi^{i},\pi^{-i}}(s_t) = \mathbb{E}_{\pi}\left[\sum_{f=0}^{\infty} \gamma^{f} r^{i}_{t+f+1} \mid s_t\right] \tag{12}$$

The above value function definition makes clear that in the special case where all agents are to be coordinated as one decision-maker then a SARL algorithm can be used. In the MARL case, the *Nash Equilibrium* solution to a game is the solution where each agent achieves its best response [53].[14] However a Nash Equilibrium is not necessarily unique [31]. The global solution is therefore a *Pareto Optimal* Nash Equilibrium [31, 54]. This refers to the scenario where a joint policy dominates all other joint policies and no other policy yields greater value.[15]

As with SARL, one can define a value-based solution method for MARL. Multi-Agent Deep Q Networks (MADQN) [55] extends DQN [8] to the cooperative multi-agent case. As the agents explore their environment they collect their trajectories of observations, actions and rewards and store these. These are then used as training data to update the parameterised Q_θ function estimate. This is usually estimated with a deep neural network. The randomised experience buffer $\mathcal{B}$, containing tuples of agent experiences, is used here in a similar manner to the single-agent DQN. Here, the Q_θ function estimates the value of taking action a_t after observing o_t, $Q_\theta(a_t, o_t)$. The neural network can train on the collected data by minimising a loss function similar to below [56]. The loss function contains a bootstrap expression as it makes use of a previous model of the Q function, Q_θ to estimate itself. MADQN minimises a squared-error objective function $L(\theta')$ similar to [56, 57]:

$$L(\theta') = \mathbb{E}_{(\mathbf{o}_t,\mathbf{a}_t,r_t,\mathbf{o}_{t+1})\sim\mathcal{B}}\left[\sum_{i=1}^{W}\left(r^{i}_{t} + \gamma \max_{a} Q_{\theta'}\left(o^{i}_{t+1}, a^{i}\right) - Q_\theta\left(o^{i}_{t}, a^{i}_{t}\right)\right)^2\right] \tag{13}$$

This provides a Q network that is a linear sum over all the agents' cumulative rewards and, therefore, is analogous to single agent Q-learning. The minimised loss function provides the new θ' parameters which the agents can then use as their policy parameters.

An analogous learning algorithm to an actor-critic method is *Multi-Agent Deep Deterministic Policy Gradients* (MADDPG) [58]. This algorithm aims to learn a policy distribution over actions, $\pi_\theta(a|o)$. Here the action most likely to maximise reward is assigned the highest probability in the distribution [56]. As this is an actor-critic method one estimates the policy-gradient as below, where ρ^π is the observation distribution [56]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{o\sim\rho^{\pi},a\sim\pi_\theta}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(a_t \mid o_t\right)\left(r_t - V\left(o_t\right)\right)\right] \tag{14}$$

Interestingly, instead of learning a Q function for each agent one can use algorithms that learn a single shared function. This is often referred to as *parameter sharing* within the MARL literature. This can yield faster and more stable training [56, 59, 60, 61]. Algorithms that use parameter sharing include *Q-Mix* [61]

[14] Interestingly, every *normal form* game has at least one Nash Equilibrium [53].

[15] A joint policy π_1 is said to Pareto-dominate a joint policy $\pi_2 \iff V^{i}_{\pi_1}(s) \geq V^{i}_{\pi_2}(s) \forall i, \forall s \in \mathcal{S}$ and $\exists j, \exists s \in \mathcal{S}$ for which $V^{j}_{\pi_1}(s) > V^{j}_{\pi_2}(s)$ [31].

and *Counterfactual Multi-Agent Policy Gradient* (COMA) [62]. On the other hand while some variants of *Multi-Agent Proximal Policy Optimization* (MAPPO) use parameter sharing, others do not [63]. These solution algorithms can be categorised in various cooperative MARL paradigms for training and execution which we will now briefly discuss.

2.1.2.3 Training and Execution Schemes and Challenges in Cooperative MARL

The number of agents in a multi-agent system affects the computational complexity of the system. In fact, as more agents are added to the system, the state-action space grows exponentially [31, 64, 65]. To deal with this computational complexity, the cooperative MARL literature has devised several schemes for the learning, training and execution of agents.

Agent training refers to their learning and collecting of experience data in order to optimise their parameters in order to know how to act to accumulate the greatest amount of reward. Training can be divided into two broad paradigms: (i) centralised and (ii) distributed [31].

Centralised training refers to the case where agents share some of their experience with one another during the updating of parameters. In contrast, distributed training refers to where agents behave as in SARL. Agents in the distributed training paradigm update their parameters based on their own experience alone.

The execution of agent policies describes the situation where agents select actions to engage with their tasks. As with training, execution can be divided into two cases. These are (i) centralised and (ii) decentralised execution [31]. The first is where agents select their actions based on a centralised unit and the second is where actions are determined according to the agent's individual policies.

These two categories give rise to the three paradigms of cooperative MARL algorithms [31]. (i) Distributed training, decentralised execution (DTDE), (ii) centralised training, decentralised execution (CTDE) and (iii) centralised training, centralised execution (CTCE). While it is interesting to analyse the traits of the different types of MARL algorithms theoretically, in this dissertation we focus only on the empirical algorithm performance evaluation. Here, however, we note some theoretical considerations, seen in the literature, of the different algorithm paradigms.

DTDE has been found to suffer from the non-stationarity of MARL environments [31]. Non-stationarity in MARL refers to how the presence of multiple agents interacting with the environment affects the ability of a single agent to view their interaction with their environment in terms of the Markov property [66, 67]. Environments in the multi-agent setting are susceptible to the joint actions of all the agents. As any one agent does not necessarily have access to knowledge of the joint actions or the states of the other agents, it becomes difficult to model an optimal policy. Non-stationarity can be overcome using centralised training structures such as a shared value function, parameter sharing or communication [31]. Learning to communicate is, itself, mentioned in Gronauer and Dieopold [31] as a challenge facing MARL. Various communication algorithms are mentioned in section 2.3.

In the CTCE paradigm a centralised executor models the joint policy that maps a joint, shared collection of observations to a set of distributions over actions [31]. While CTCE is conducive to the use of SARL methods, the state-action space grows exponentially with the number of agents. To combat this (*curse of dimensionality*), papers like Gupta et al. [59] have represented the centralised executor as a set of independent policies. This assumes independence between the action distributions of the agents. Agent's individual policies are captured independently and combined in a factorised form to yield the joint policy distribution such that $\pi = \prod_i \pi(a^i|s^i)$ [31]. The corresponding action value function is then a sum of the independent value functions such that $Q_\pi = \sum_i Q_{\pi^i}(o^i, a^i)$. The CTCE paradigm may yield the problem of *lazy agents* whereby some agents may learn a good policy but this may cause another agent to have less incentive to do so [68].

One can also employ an independent learner algorithm and ignore the existence of the other agents [54]. However, according to Lowe et al. [58] and Matignon et al. [54] independent learners perform poorly in stochastic and complex environments and, according to Lanctot et al. [69], they can over-fit at training time and fail to generalise at execution. As mentioned, another problem with independent learners is that of scalability. As the number of agents in a multi-agent system grows, so too does the computational complexity [64, 65]. Foerster et al. [62] note that distributed training for actor-critic methods is slower than when centralised. Gupta et al. [59] suggest that methods that are trained in a distributed manner perform worse than their centralised training counterparts. These works suggest a poor scaling of DTDE with the number of agents [31].

The CTDE paradigm has agents learn their respective individual policies using extra pieces of information from other agents [31]. At execution time this extra information is not available to them. CTDE has been used to overcome non-stationarity through the passing of information about other agents' actions at training time. Training can be sped up (relative to DTDE) through the process of sharing parameters, observations or actions [31, 59, 62]. Shared, centralised value functions can also be used as a shared critic that is trained using data collected by multiple agents in the CTDE paradigm [58]. Various ways of decomposing such a value function can assist with credit assignment whereby each agent may face a simplified sub-problem to solve [31, 61, 68, 70, 71].

Other than DTDE, CTDE and CTCE, various MARL communication algorithms allow for a fluidity of categorisation between CTDE and CTCE due to the sharing and flow of various information between agents. This is sometimes done in a networked fashion such as with the *CommNet* algorithm [72]. This alludes to the problem of coordinating and communicating between agents.

Separately to communication, coordination refers to the challenge of how agents may cooperate to achieve a joint goal [31]. Appropriately coordinating agents may require correct credit assignment which refers to correctly rewarding agents in proportion to their shared contribution to the achievement of the joint reward. Methods of value decomposition may achieve this [61, 68, 73]. Communication and coordination can be used to overcome the challenge of partial observability in the agents' tasks whereby agents may only be aware of partial information as to the current state of their environment.

While each of these challenges poses an interesting theoretical problem for cooperative MARL algorithms to tackle, and while theoretical analysis of the many algorithm paradigms is both interesting and valuable, in this dissertation we turn (only) to the problem of algorithm evaluation in terms of empirical performance of one algorithm relative to one another.

In the next section we define and discuss exactly this form of evaluation. We review the RL literature for commentary on RL evaluation and attempt to demonstrate why this remains a challenge facing RL research. This effort will involve systematically understanding the components that make up RL evaluation and attempting to answer our first research question on identifying the specific problems with RL evaluation.

2.2 Evaluation in RL

In an attempt to understand the level of rigour in the evaluation of empirical algorithm performances in the RL field, we first survey the literature. The RL field has seen increased research that highlights the methodological flaws in empirical algorithm performance evaluations [16, 17, 18, 19, 20, 21, 22, 23, 74]. Here, we study this literature with a view to, later, drawing upon this literature set's recommendations when attempting to construct a better guideline for RL evaluation. More specifically, our literature survey here will reveal several concerning aspects about the level of rigour associated with RL algorithm performance evaluations. It will then turn to how these concerns have been addressed, specifically addressing the lack of experimental protocol for deep RL algorithm evaluations. This section aims to systematically understand the components of RL algorithm performance evaluations. We then aim to address our first research question and uncover the specific problems with this.

In RL, the use of a small number of algorithm evaluation runs (Figure 4) can easily mean that two different trial evaluations can yield significantly different empirical algorithm performance distributions. Furthermore, algorithm performance distributions can vary based on the use of different random seeds, hyperparameter choices or implementation details [18, 19, 21, 22, 23]. The imperative of taking statistically meaningful performance measurements is therefore a challenge [19, 23].

As a result of this challenge it is not uncommon to observe variance in reported algorithm performances (for the same algorithm) across different research papers [1]. Additionally, the study of the true source of performance improvements (from algorithm development) for any particular algorithm is often neglected. A particular issue observed is that the source of such performance gains is often yielded from code-level optimisations rather than the development of novel algorithm features [22, 75]. Indeed, a focus is placed in the RL literature on, merely, improving performance measurements for an algorithm, rather than obtaining a scientifically rigorous understanding for *why it is* that an algorithm's performance has improved.

The literature, surveyed below, offers attempts at providing more meaningful measurements of RL algorithm performance. The survey addresses some of the aforementioned challenges by stressing the imperatives of measuring statistical uncertainty [19, 23]. It also stresses the need to measure performances over multiple RL tasks [12, 23]. In the next few subsections we review these ideas along with all key aspects of algorithm evaluation in RL.

We review the selection of RL environments which act as a test-bed for the RL agents to be evaluated in. We then review the choices of measurement and aggregation for the RL algorithms' performances. This is matched with how uncertainty is measured and portrayed in the literature. Lastly, we briefly touch on how results are reported and how variance in reported algorithm performances can result from differences in implementation and evaluation procedures between research papers. Particular attention is paid to the normalisation function. This is used to normalise RL algorithm performance scores across RL tasks which may have different reward scales. In response to our first research question on what specific problems may arise with RL evaluation, we hypothesise a problem with the normalisation function as being one of these. As we will see this is due to its failure to scale performance scores with the level of skill of a particular algorithm [21].

Overall, the challenges with RL evaluation, surveyed in this literature review, serve to motivate for this dissertation's key endeavour: The suggestion of an improved protocol for empirical RL performance evaluations. The insights drawn in this section are carried forward toward the development of such an improved guideline (which we discuss in section 3).

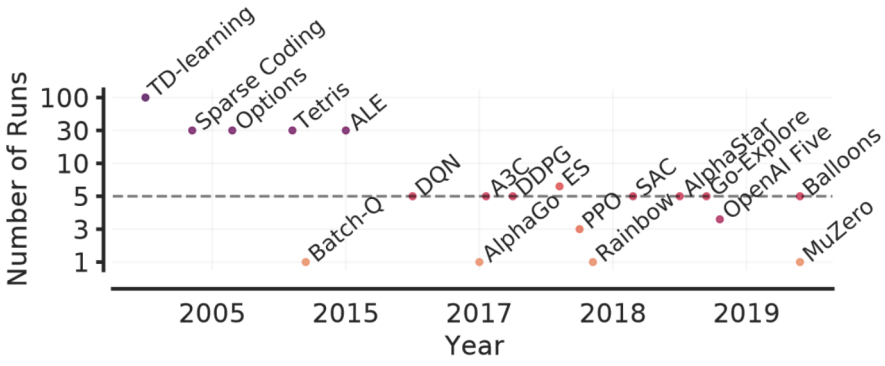

Figure 4: Figure from Agarwal et al. [23], depicting the use of very few runs in papers claiming empirical results for deep RL algorithms. Between $3-10$ runs are prevalent due to computational overheads in training and evaluating deep RL algorithms.

2.2.1 RL Task Selection For Evaluation

The first concern we address as an important aspect of the scientific evaluation of RL algorithms is the choice of environments or tasks in which to evaluate the algorithms one is studying. As mentioned above, RL algorithms are sensitive to their various settings, parameterisations and hyperparameter choices. Therefore the choice of environment that an algorithm is trained on is significant as this presents the scenario in which parameters, hyperparameters and settings are tuned.[16] Jordan et al. [21] suggest that for an RL algorithm to meet their requirement to be *usable*, its performance should be measured over a wide range of tasks.[17] [18]

As with supervised learning, exploiting known aspects of the data[19] that an algorithm is trained on, can be exploited to the benefit of the algorithm's measured in-sample performance. For instance Zhang et al. [74] and Machado et al. [14] note that, as the Arcade Learning Environment (ALE) [27] is deterministic, it is possible for deep RL algorithms to exploit their capacity to memorise trajectories and achieve high scores through *overfitting*. Defining this, we say that RL algorithms can be said to have overfit their environments when an algorithm performs well on that environment (or task) but does not perform as well over a distribution of tasks that it is designed to act in [12].

The risk of overfitting in ML is presented by learning with high-capacity models and long training times on powerful hardware [74, 76, 77]. The trade-off in ML between unleashing the capacity of these resources and regularising the training to prevent overfitting is key to out-of-sample generalisation [74]. Overfitting can therefore be seen to correspond to the memorisation of sampled data, which corresponds to poor generalisation [16]. More specifically, in supervised learning, we assume an unknown data distribution, $\mathcal{D}$, from which independent and identically distributed samples are drawn, $\mathcal{X}_{train}$ with some labels $\mathcal{Y}_{train}$ with individual elements $y \in \mathcal{Y}$ [16]. The goal is to learn a mapping $f : \mathcal{X}_{train} \longrightarrow \mathcal{Y}_{train}$. The learning of f often corresponds to the minimisation of some loss function $L(f(x), y)$. We can define the *generalisation error* e_g as [16]:

$$e_g = \mathbb{E}_{x,y\sim\mathcal{D}}\left[L(f(x), y\right] - \frac{1}{N}\sum_{i=1}^{N} L\left(f\left(x_{train,i}\right), y_{train,i}\right) \tag{15}$$

Equation 15, above, reflects the difference between a true and expected error, where a large e_g corresponds to overfitting and poor generalisation [16]. Here we have $N = |\mathcal{X}_{train}| = |\mathcal{Y}_{train}|$. However, as we do not have access to the true joint data generating distribution of x and y we instead calculate a proxy for the expected error using a subset of the available data called the test set which has $\mathcal{X}_{test}$ with some labels $\mathcal{Y}_{test}$. This has $M = |\mathcal{X}_{test}| = |\mathcal{Y}_{test}|$ such that [16]:

$$e_g = \frac{1}{M}\sum_{i=1}^{M} L\left(f\left(x_{test,i}\right), y_{test,i}\right) - \frac{1}{N}\sum_{i=1}^{N} L\left(f\left(x_{train,i}\right), y_{train,i}\right) \tag{16}$$

Extending Equation 16 to the RL evaluation domain, we note that data is generated by the RL tasks that an agent acts in. The feedback signals are generated from the reward signal, which the agent wants to maximise. Hence we replace the loss function in the expression above with the reward function r. Here, the difference between the training data and testing data can be achieved by separating random seeds for

[16] by hand or otherwise.

[17] In contrast, Colas et al. [19] and Henderson et al. [17] provide guidelines for statistical significance testing when comparing algorithms on only a single RL task [23].

[18] Within the context of the recommendation by Jordan et al. [21], it would still be relevant to test ones algorithm over several tasks which are only relevant to one's specific engineering problem. Additionally, showing an algorithm to perform well on a specified, particular subset of tasks and not on other subsets may be insightful.

[19] and benchmark environment.

instance where $\mathcal{S}_{0,train}$ and $\mathcal{S}_{0,test}$ are different sets of initial states such that $s_{train,i} \in \mathcal{S}_{0,train}, s_{test,i} \in \mathcal{S}_{0,test}$ and T is the length of each episode [16]. This is expressed below [16]:

$$e_{g,RL} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} r\left(s_t, \pi_\theta\left(a_t|s_t\right) \mid s_0 = s_{train,i}\right) - \frac{1}{M}\sum_{i=1}^{M}\sum_{t=1}^{T} r\left(s_t, \pi_\theta\left(a_t|s_t\right) \mid s_0 = s_{test,i}\right) \tag{17}$$

Other differences between training and test sets which would affect Equation 17 may include changes to transition dynamics, reward functions, or to the initial state distributions which are expressed above [16].

Practically, experimental protocols like *cross-validation* can be used to verify the prevention of overfitting in the supervised learning case [74]. Such protocols, however, are a challenge in the RL setting due to the computationally heavy demands of running RL algorithms in practise. Unlike supervised learning, in practice, the setup of RL experiments usually omits explicitly separated training and testing stages [74]. However, as with supervised learning, the performance of an agent on a training task often does not reflect its performance on an unseen task. Hence we see a lack of strict experimental protocols for the validation of the prevention of overfitting in the RL case.[20] Hence we turn to the advice of Whiteson et al. [12] and Agarwal et al. [23].

To combat environment overfitting, Whiteson et al. [12] recommend using their *generalised evaluation methodology*. Here, tasks are sampled from some general set of tasks for training an algorithm.[21] Subsequently, a second sample is used as a testing set to test the trained algorithm. This approach is echoed by Jordan et al. [21] and the Procgen benchmark [78][22] and also mimics supervised learning.

Whiteson et al. [12] define the *generalised environment* as $\mathcal{K} = \{\Theta, \mu\}$, where μ is a distribution over some set of environments Θ. Here, Θ allows for the specification of a target class of environments and μ specifies the relative importance of each environment within that target class. In this setting, an RL algorithm models not only a single MDP but also models uncertainty across the possible MDPs it can be exposed to. This prevents algorithms from overfitting to test environments as the designer of the algorithm is unaware of the test environments that will be sampled. However, with knowledge of μ, there remains the potential for *uncertainty overfitting* where the designer customises an algorithm to overfit the distribution over the generalised set. A solution to this, proposed by Whiteson et al. [12], is to have some *secret generalised* set for evaluation. Unfortunately, this method has practical limitations as the distribution of the test set must be kept secret and would only be practical for competitive one-shot challenges. Alternatively, Machado et al. [14] suggest that a fixed split between the training and test sets be maintained for benchmark sets with a wide array of scenarios, such as ALE [27].

An alternative to maintaining a secret generalised set is to use a *meta-generalised* methodology where the algorithm's performance is aggregated across multiple *meta-trials* [12]. For this method, the distribution selecting the environments from the generalised set is changed between each trial which prevents the agent from learning the sampling distribution. Although more practical than a secret generalised set, this method of evaluation is more computationally expensive and may not be time-efficient for complex environments.

Agarwal et al. [23] note that computationally expensive tasks such as the ALE [27] lead to the measurement of performance being difficult in deep RL. This is largely due to only a small number of runs per task being available because of computational cost. This means that providing statistically robust estimates of performance is difficult. As computational tractability is a key aspect of evaluation in RL, as noted in Jordan et al. [21], Agarwal et al. [23] set to robustly measure empirical algorithm performance when provided with only a *handful of runs* per task (5-10 runs per task). Both Agarwal et al. [23] and Jordan et al.

[20] Moreover, combined with a lack of standardised performance measures, the robust measurement of RL algorithm performances is elusive.

[21] Whiteson et al. [12] note that a target distribution can be specified as some more narrow distribution if the task that the algorithm is being designed for is more specifically defined.

[22] Procgen [78] implicitly creates a distribution of tasks to sample from using a procedural generation process.

[21] advocate for the aggregation of an empirical algorithm performance measurement to be computed over multiple tasks whilst maintaining computational tractability.[23] While this aggregation is complicated by the need to normalise RL algorithms' scores, we will address a proposed resolution to this using multi-criteria decision analysis (MCDA) in section 2.4.

Significantly, the prevailing consensus in the RL evaluation literature is the use of multiple RL tasks for the empirical performance evaluations of RL algorithms [12, 21, 23]. Whilst we advocate for the use of multiple tasks, we also note that the set of tasks selected for training and testing one's algorithms should be relevant to the engineering context that one is researching.[24] If one were only interested in a narrow context of tasks then some variation of those should be used, for example, as the set of environments Θ proposed by Whiteson et al. [12]. Carrying this recommendation forward we now turn, in the next subsections, to the *measurement* of algorithm performances and the measurement of uncertainty.

2.2.2 Measurement of Uncertainty in Algorithm Performance

Algorithm performances in RL have many sources of variation. For instance, the selection of different hyperparameters or random seeds when an algorithm is initialised can lead to a highly variable set of reported results. It is therefore essential to understand and correctly report uncertainty [19, 23]. However, the evaluation literature notes that the reporting of uncertainty in RL has seen inconsistency, inaccuracy or been ignored altogether [17, 18, 19, 21, 23]. The literature notes that some publications report only point estimates such as the mean or median of performances *without* reporting uncertainty at all [17, 23]. The problem of not reporting uncertainty is then exacerbated since runs of an RL algorithm are computationally expensive and hence it is common for only a few runs of an algorithm to be completed before reporting results. Furthermore, the highly variable results mean that the reporting of only a point-estimate from a small sample provides a depiction of the result that may be an outlier. Therefore, some literature recommends increasing the number of runs to accumulate a better sample [21]. However we take the view of Agarwal et al. [23], that computational limits render this option impractical.

As is recommended in Agarwal et al. [23], one can mitigate the effects of having a small sample via the use of bootstrapping [79]. Specifically, stratified bootstrap confidence intervals[25] can be used as an effective means of conveying uncertainty in RL [23].

To compute these stratified bootstrap confidence intervals, the set of algorithm performance score data can be sampled from (with replacement) [1, 23]. Here, the set to be sampled from has $N \times M$ data points, corresponding to the number of algorithm runs per task, N, and to the number of tasks, M, that the algorithms are built to perform in.[26] Several repeated samples are collected from the $N \times M$ data points. For each one of these samples, a sample statistic is calculated which provides an estimate of the true score value we are interested in. The distribution of these sample statistics provides the sampling distribution of the statistic in question. From this distribution, the uncertainty bootstrap confidence intervals[27] can be estimated.

[23] According to Agarwal et al. [23], the aggregation over tasks mean that fewer runs per task are necessary to compute the desired aggregate algorithm performance metrics.

[24] We do not address the problem of generalisation in this dissertation.

[25] A 95% confidence interval can be interpreted for a finite-sample score as an estimate for the true value of that score [23]. Here, if the true score lies outside the interval then a sampling event with probability of less than or equal to 0.05 has occurred in the measurement of the interval. Bootstrapping refers to the repeated use and sampling of the same set of data and, lastly, *stratified* refers to the data originating from different environments or tasks.

[26] Tasks are sometimes referred to as environments or scenarios in the literature [80]. In this text we refer to an environment as a collection of one or more tasks. For instance, The StarCraft Multi-Agent Challenge (SMAC) [81] or Multi-agent Particle Environment (MPE) [58] environments contain many tasks that an algorithm can be run on.

[27] An $\alpha \times 100\%$ confidence interval can be interpreted such that if the experiment in question is rerun, and a confidence interval is constructed using the different sets of runs each time, the proportion of intervals that contain the true score value will tend toward α [23]. Here, $\alpha \in [0, 1]$ is the nominal coverage rate of the confidence interval.

Agarwal et al. [23] calculate the bootstrap confidence intervals using 50000 bootstrap samples to compute the sampling distribution for the score[28]. This distribution is used to calculate the confidence interval using the percentile bootstrap method, which calculates the $1-\alpha/2$ and $\alpha/2$ quantiles to yield the interval for the score, X as $\left(x^*_{(\alpha/2)}, x^*_{(1-\alpha/2)}\right)$.[29] These are calculated empirically but are defined such that given the cumulative distribution function (CDF) for the algorithm's performance score, X, $F_X : \mathbb{R} \to [0,1]$ with $F_X(x) := \Pr(X \leq x)$, the quantile function retrieves the value of x that corresponds to the α quantile. Here the quantile function can be seen as the inverse CDF, $Q(\alpha) = F_X^{-1}(\alpha) : [0,1] \to \mathbb{R}$ having $Q_X(\alpha) := \inf_x \{x \in \mathbb{R} \mid F_X(x) \geq \alpha\}$ such that $x^*_{(\alpha/2)} = Q_X(\alpha/2)$. Percentile intervals are found to provide good interval estimates for as few as $N = 10$ runs [23].

Confidence intervals are widely advocated for, for the purpose of measuring uncertainty and estimating effect sizes [23, 82, 83, 84]. Agarwal et al. [23] contrast the use of intervals with statistical significance tests, noting that such tests can be misinterpreted [85, 86, 87].[30] This contrasts with other work in the RL evaluation field such as that of Colas et al. [19] which reviews statistical tests for meaningful comparisons between RL algorithms.

In addition to preferring interval estimates over statistical tests, we note that the work done by Colas et al. [19] relies on a high number of runs, 20-30 runs per task, as opposed to the 5-10 runs per task advocated for by Agarwal et al. [23]. In a separate paper, Colas et al. [18], discuss the correct number of random seeds to use to attain a statistically significant comparison of performance between two algorithms. This work is applicable to statistical comparisons on a single task only. Whilst the methodology here is sound, with the use of statistical power analysis, we again take the view of Agarwal et al. [23] that, in the RL field, only a handful of runs are tractable. Moreover, only a handful of runs are required with the use of stratified aggregation over tasks. Furthermore, we also take Agarwal et al. [23]'s view against the deference to statistical significance testing due to the dichotomous nature of such tests.

Interestingly, Colas et al. [19] distinguishes between evaluation during training and evaluation after training.[31] This distinction is also made in Chan et al. [20]. In their work another aspect of uncertainty measurement is discussed in terms of proposed metrics for measuring the reliability of RL algorithms.

Table 1 demonstrates the reliability metrics as proposed and displayed by Chan et al. [20].[32] The metrics focus on measuring the reliability and robustness of RL algorithms, via the measurement of variability, both during training and after training. The metrics are designed alongside complementary statistical tests to enable rigorous comparisons. The metrics provide a mechanism to reveal aspects of algorithms that are obscured when only measures of central tendency (such as mean or median) are provided.

The metrics found in Chan et al. [20] measure variability *across* training runs and variability *across* rollouts of a fixed policy after training. It is these two aspects which they refer to as *reproducibility*. The metrics also measure what they refer to as *stability*. This is the variability *within* training runs. Stability is measured by the metrics on the top row of Table 1, while reproducibility is measured by the bottom two rows. In addition to stability and reproducibility, the metrics by [20] capture dispersion and risk. Here, by dispersion, they are referring to the width of distributions of returns. Risk is referring to the heaviness and extremity of the lower tail of distributions of returns. Importantly, the *variability across runs* allows for the capturing of the algorithms' sensitivity to random seeds, hyperparameter settings and implementation details.

[28] They use 2000 samples for point-wise confidence bands and the average probability of improvement.

[29] Agarwal et al. [23] compare the percentile interval to the basic bootstrap interval, the bias corrected interval and the bias-corrected and accelerated interval. In their study they find that the percentile intervals have the best coverage, in terms of covering the true parameter, as well as the smallest interval width as compared to the other three intervals tested.

[30] As statistical significance tests are of a dichotomous nature, whereby they contrast significant vs non significant results, their results can be misinterpreted. For instance, results which are of practical significance can be deemed non-significant from the tests. Furthermore, results which are not practically significant can yield significant results from the tests [23].

[31] where evaluation after training occurs on a fixed policy after learning [20].

[32] Interestingly, these metrics appear in an earlier version of Papoudakis et al. [88] which is found in [89].

Measurement for evaluation during training is made, in Chan et al. [20], on the input of performance curves of an algorithm.[33] Measurement after training is made via the input of performance scores of a set of roll-outs of a fixed policy. These measurements are discussed in the next subsection.

As a measure of dispersion, the inter-quartile range ($\text{IQR} = \theta^*_{(0.75)} - \theta^*_{(0.25)}$) is chosen by Chan et al. [20].[34] As a measurement of risk, the conditional value at risk (CVaR) is used. Here, for some random variable X and a given quantile α, with α-quantile $\text{VaR}_\alpha(X)$, Chan et al. [20] gives:

$$\text{CVaR}_\alpha(X) = \mathbb{E}\left[X \mid X \leq \text{VaR}_\alpha(X)\right] \tag{18}$$

Table 1: This table is found in Chan et al. [20]. It displays their reliability metrics as described in the main text. IQR refers to the inter-quartile range and CVaR is the conditional value at risk. Evaluations *During Training* take as input the training curves of an algorithm to measure the reliability over the course of training. Evaluations *After Training* take as input the performance scores of a set of roll-outs of a trained, fixed policy. This measures the reliability of a algorithm that has been trained.

		Dispersion (D)	**Risk (R)**
During Training	**Across Time (T)** (within training runs)	IQR within windows, after detrending	Short-term: CVaR on first-order differences Long-term: CVaR on drawdown
During Training	**Across Runs (R)**	IQR across training runs, after low-pass filtering.	CVaR across runs
After Training	**Across rollouts on a Fixed Policy (F)**	IQR across rollouts for a fixed policy	CVaR across rollouts for a fixed policy

The metrics chosen are intended by Chan et al. [20] to be robust and make minimal assumptions about the distribution of results. The authors, however, recommend the use of these metrics to be on a per task basis only. This is because algorithms can have different patterns of reliability for different tasks [20].

In contrast, the recommendations of stratified bootstrap confidence intervals by Agarwal et al. [23] offer uncertainty measurements across tasks. However, these stratified bootstrap confidence intervals specifically pertain to measuring uncertainty after training. On the other hand, the recommendations by Chan et al. [20] offer evaluators methods to measure reliability during training and between training runs. While, as a proof-of-concept demonstration on the OpenAI Gym [90] environment, Chan et al. [20] measure the reliability of five RL algorithms using 30 runs per task, they also demonstrate on ALE [14] an effective use of their metrics using only 5 runs per task for four algorithms.

Recommendations for bootstrap confidence intervals in works by Henderson et al. [17] and Colas et al. [18, 19] require at least $20 - 30$ runs per task [23]. Moreover these works pertain to measurement on only a single task. Hence this dissertation places focus on the stratified aggregation of Agarwal et al. [23] which makes provision for computationally tractable uncertainty measurements, using only a *handful of runs* per task. Taking this recommendation along with the recommendation to make use of multiple environments for evaluation, the next subsection focuses on how to measure the raw data upon which to estimate uncertainty. We will then examine how to aggregate this data and the problems that arise therein.

[33] See Figure 5 for an example of RL algorithm training performance curves.

[34] Chan et al. [20] note that the IQR does not require an assumption of Gaussian data and is preferred to the median absolute deviation from the median as the IQR is more suitable for asymmetric distributions. The authors also note IQR is a more robust statistic than variance.

2.2.3 Empirical Algorithm Performance Measurement

As has been discussed in the above subsections, variability and inconsistency in reporting and evaluation procedures plague the field of RL. We have discussed how the choice of environments is a vital step in the evaluation procedure and how a lack of strict experimental protocols and a lack of standardised performance measures exist [12, 14, 17, 21, 23, 74, 27]. We have begun investigating approaches to an evaluation protocol by looking at the measurement of uncertainty whilst acknowledging the sparsity of data due to computational constraints [23]. The next step is to examine the empirical *measurement* of the performances of deep RL algorithms.

Jordan et al. [21] notes that it is the use of flawed metrics for algorithm performance measurement that leads to inconsistencies in reporting. Their work defines a new procedure for evaluating SARL algorithms over multiple tasks. They focus on comparing algorithm performances on a range of tasks *with little or no task-specific tuning*. As noted in Agarwal et al. [23], the recommendations here require evaluating over 1000 runs of each algorithm per task to obtain significant results. As hyperparameter tuning is a common and accepted part of the engineering of RL algorithms, we take the view of Agarwal et al. [23] that we should focus on reliably measuring RL algorithm performances *after* hyperparameter tuning with only a handful of runs to maintain computational tractability.

On the other hand, many recommendations for the evaluation procedure in Jordan et al. [21] remain valuable. They identify four properties which make an evaluation procedure useful. These are that an evaluation procedure should have the following properties [21]:

- Scientific: The procedure should answer a research question, test hypotheses and quantify uncertainty.
- Computationally tractable: The procedure should be such that a researcher should be able to run and repeat experiments without too great a computational overhead.
- Usable: Performance measurements should quantify an algorithm's performance over a wide array of tasks. This will therefore quantify its *usability* over such a range of tasks. Additionally, the metrics used should quantify the usability of the algorithm in terms of its training time and time spent tuning hyperparameters.
- Non-exploitative: The procedure should not allow an algorithm to be favoured because it performs well on an over-represented subset of tasks, abuses a score normalisation method or overfits an environment.

Hence, while we have addressed the computational tractability requirement and the requirement for an evaluation procedure to be non-exploitative in terms of environment overfitting, our scope of a usable, scientific and non-exploitative evaluation measurement procedure remains incomplete. More specifically we have found that Agarwal et al. [23] tackle the computational tractability requirement in a compelling manner by addressing performance evaluations with only a handful of runs. Whiteson et al. [12] address the non-exploitative requirement by suggesting the use of multiple environments and by recommending methodologies for selecting a set of tasks for evaluation. We have also addressed the scientific requirement insofar as to recommend the use of stratified bootstrap confidence intervals for the quantification of uncertainty [23]. What remains is to address the measurement of performance data in a manner which meets the aforementioned requirements.

Following Jordan et al. [21], the evaluation of RL algorithms can be divided into three phases. These are (i) data collection, (ii) data aggregation with the quantification of uncertainty and lastly, (iii) displaying results. These constitute the next few subsections.

2.2.3.1 Data Collection

Jordan et al. [21] note that most approaches to data collection consist of two phases. First, hyperparameters are tuned and second, the best hyperparameters are selected and executed for a number of trials. As Jordan et al. [21] desire to satisfy (in their evaluations) the requirement that algorithms be *usable* and therefore have a limited training time and hyperparameter tuning time, they object to this method of data collection. They note that this method neglects the amount of data used for hyperparameter tuning. Moreover, an algorithm that requires a lot of tuning could be favoured to one that requires little tuning. The tuning has no measurable consequence and will likely benefit the first algorithm in a performance comparison with the second.

Jordan et al. [21] also note that this method violates their scientific property requirement. This is since the measurement of the sampled algorithm performances, after hyperparameter tuning, reflects only the distribution of scores for that particular hyperparameter configuration. Statistical claims are therefore only pertinent to the particular hyperparameter configuration being evaluated. To resolve this dilemma, Jordan et al. [21] define a *complete algorithm*. This is an algorithm where the only required input is the meta-information about its tasks such as its state-action space. In their work, algorithms are made complete by sampling hyperparameters randomly from several hyperparameter distributions.

Addressing the problem of hyperparameter tuning and data collection, we note that different choices of hyperparameters can have significantly different effects on algorithm performances [17]. Because of this, many algorithms' intended performances can suffer without tuning. Hence papers such as Agarwal et al. [23] adopt an alternative view of hyperparameter tuning to the one proposed by Jordan et al. [21]. More specifically, Agarwal et al. [23] adopt the view that focus should turn to the reliable measurement of RL algorithm performance after hyperparameter tuning.

To resolve the question of how to select hyperparameters for a scientific, statistical comparison of algorithms, Henderson et al. [17] recommend selecting hyperparameters so that the algorithm's performance matches its original reported performance.[35] This implores upon the evaluator the necessity to report all hyperparameter choices and provide the original code-base used to generate results [17, 2]. This would also necessitate the creation of some universal test-bed of tasks for the algorithms, or the re-use of tasks that were used to test the original algorithm versions. As new tasks and environments emerge regularly, and as no universal testing suite exists, we may need another method of hyperparameter selection. Hence a second alternative is to make the best attempt to tune hyperparameters to their best possible performance [13]. This offers the opportunity for fair comparison between algorithms. Ultimately the resulting statistical comparisons will pertain only to the chosen set of hyperparameters.

With a selection method for hyperparameters in mind, we turn to the problem of what specific measurements of data to collect. Distinction can be drawn between evaluation during training and evaluation after training [19, 20]. According to Colas et al. [19], a measurement of an RL algorithm's performance after t training steps should be evaluated as the average over $E \in \mathbb{N}$ evaluation episodes. These are conducted independently from training using the current version of the policy at time t. In other words, this is a roll-out of a fixed policy at time t [20]. It is up to the evaluator to decide on the duration of training, denoted by the number of timesteps $T \in \mathbb{N}$). They must also determine how frequently to evaluate the fixed policy roll-out averages, choosing specific timesteps t at which to take the averages. Lastly, the evaluator must decide on the number of evaluation episodes E to average over at every evaluation interval.[36] Plotting the averages over time corresponds to plotting an algorithm's learning curve over time. These are also referred to as training curves or sample efficiency curves [1, 23]. An example of some algorithm training curves can be seen in Figure 5.

[35] unless of course this is the first time that the algorithm is being used.

[36] Gorsane et al. [1] find that it is most common in MARL for algorithms to be trained for $T = 2$ million timesteps with evaluation averages being taken every $t^* = 10000$ timesteps. They also find that the most common number of evaluation runs in MARL is $E = 32$.

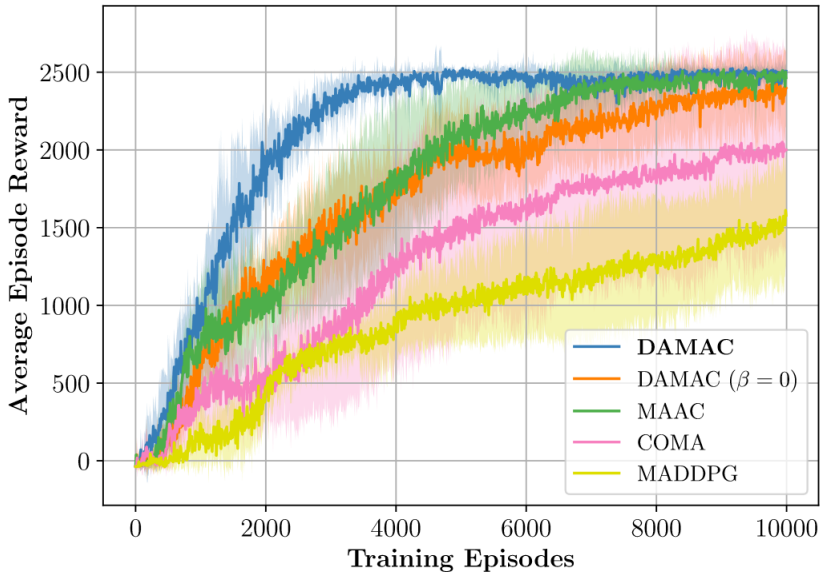

Figure 5: An example of five RL algorithms' training curves from the results of Saeed et al. [91]. RL algorithm training curves are also referred to as sample efficiency curves or learning curves. This is since they demonstrate the training or learning or RL algorithms over time (number of training episodes in the above). This is also since they demonstrate how quickly an RL algorithm can converge to optimal performance (reflected as 2500 average episode reward in the above) and therefore, how sample efficient the algorithm is (if it requires fewer training episodes to reach optimal performance, it is more efficient). The five algorithms whose training is seen above are DAMAC, DAMAC ($\beta = 0$), MAAC, COMA and MADDPG. Ideally, one plots a training curve by deciding on the duration of training $T \in \mathbb{N}$, and then plotting every $t^* \in \mathbb{N}$ timesteps the mean of $E \in \mathbb{N}$ roll-outs of the fixed policy at that particular time t (E evaluation episodes).

In contrast to Colas et al. [19], the training curves plotted in Chan et al. [20], for instance, are plotted using the average returns over recent training episodes which are collected across time using a rolling window, and using the returns of the policy as it evolves over time.[37] [38] Colas et al. [19] note that the common practice of evaluating algorithms by averaging over several training episodes (over time) results in far less interpretable performance measurements as compared to taking averages of the E evaluation episodes of the check-pointed policies.

Chan et al. [20] do, however, choose to evaluate over $E = 30$ evaluation episodes to measure performance after training.[39] They choose to do this using the last check-pointed policy at the end of training. Colas et al. [92] refer to this as the *final metric*. This is also used by Agarwal et al. [23]. Henderson et al. [17] and Machado et al. [14] use a version of the final metric which averages over the n last training episodes. Colas et al. [92] suggest that this is subject to the same criticism as given by Colas et al. [19], that this results in a far less interpretable performance measurement than averaging over E evaluation episodes with a fixed policy. Furthermore, the final policy at the end of the prescribed training time may not have converged, as is often the case, to the best policy. Hence, Colas et al. [92] recommend using the *absolute metric*. They define the absolute metric to be the average performance of the check-pointed best policy over $E = 100$ evaluation episodes [92]. The absolute metric is recommended in Gorsane et al. [1].

On the other hand, Agarwal et al. [23] suggest that selecting the best policy as a representation of an algorithm's performance results in a positive bias. The authors say that reported algorithm performances

[37] This method is recommended for evaluations on ALE by its authors, Machado et al. [14] [20].

[38] For their training curves, Chan et al. [20] use an evaluation frequency of $t^* = 1000$ steps with an evaluation duration of $T = 2$ million timesteps for each of the $N = 30$ algorithm runs on OpenAI Gym [90]. On the ALE [14], they use an evaluation frequency of $t^* = 1$ million frames and an evaluation duration of $T = 200$ million frames for each of the $N = 5$ algorithm runs.

[39] for their demonstration on OpenAI Gym [90].

that are produced as a result of selecting the best possible policy (and therefore taking the maximum over averaged evaluation episodes across time) are incomparable with end-of-training performances. Hence they recommend the final metric.

Thus, having selected hyperparameters, plotted training curves and attained final metrics using an average of E evaluation episodes at the end of training we have a data set of algorithm performances after training. This data set consists of the final metrics of N runs of an algorithm on some task. Recall that recommendations from Whiteson et al. [12], Jordan et al. [21] and Agarwal et al. [23] suggested that the algorithms be compared over multiple tasks. Hence we obtain N final metrics for each algorithm for some M tasks. Agarwal et al. [23] have also proposed aggregating over the tasks in order to require fewer runs per task and maintain statistical integrity. Hence we turn to the problem of how to aggregate across tasks, noting that reward scales are not necessarily equal.

2.2.3.2 Data Aggregation

Having studied what the literature has to say about data collection, we are left with a clear set of recommendations for each component of this phase of evaluation. The data aggregation phase, however, leaves us with two problems that may answer our first research question on what specific problems still need addressing in RL evaluation.

The data aggregation phase addresses two problems, according to Jordan et al. [21]. Firstly, performance data needs to be normalised to a common scale. Different tasks may have different reward functions and ranges of scores. Since we seek to aggregate over these, we need them to be on an additively compatible scale. Secondly, using a uniform weighting when aggregating across tasks can introduce bias. For instance, with a uniform weighting, when several variants of one task are included in Θ then that task is deemed more significant to an algorithm's performance than, say, another task which has only one variant in Θ [21]. While, these problems will later be addressed with multi-criteria decision analysis (MCDA), here we study Jordan et al. [21]'s approach.

For tasks $j \in \Theta$, a normalisation function $g(x, j) : \mathbb{R} \times \Theta \to \mathbb{R}$ maps algorithm performance scores x on tasks j to $g(x, j)$, which has some common scale for all tasks in Θ [21]. One common normalisation technique, discussed in Jordan et al. [21], is the performance ratio of two algorithms k and l. This is $\mathbb{E}[X_{l,j}] / \mathbb{E}[X_{k,j}]$. This ratio is not suitable as a method of normalisation, as it is sensitive to both location and scale of performance, and all changes in performance are assumed to be equally challenging [21]. For instance, a task with scores in the range $[0, 1]$ will produce larger differences in performance scores than a task with scores in the range $[1000, 1001]$ [21]. Furthermore, in response to a change in the algorithm in the denominator, this ratio can yield different (arbitrary) rankings of algorithms when using the arithmetic mean to aggregate across tasks [21, 93].[40]

Another common normalisation technique is to map scores that fall within a range $[a_j, b_j]$ to $[0, 1]$ using $g(x, j) = \frac{x - a_j}{b_j - a_j}$ for some task j [27]. However, this results in scores being incomparable between tasks, as normalised scores can cluster in different regions of [0,1] [21]. For instance, where task 1 has a score range of $[-100, 1]$ with a random agent scoring around 0, and task 2 has a score range of $[10, 1000]$ with a random agent scoring 20, the first environment will have algorithms scoring near 1 and the second will have algorithms scoring near 0 [21].

A second version of this approach, proposed in Bellemare et al. [27], is to normalise using some minimum and maximum reference scores attained via sampling algorithm (or human[41]) performances. Given recorded

[40] While the use of the geometric mean solves the issue of the changing ranks in performance [93], it does not solve the issues of sensitivity to location and scale or the issue of assuming changes in performance are of equal difficulty [21].

[41] In Mnih et al. [35], a professional human games tester is used as the maximum performance and a random agent as the lower bound.

average scores $x_{k,j}$ for algorithms k on task j: $g(x,j) = \frac{x - \min_k x_{k,j}}{\max_k x_{k,j} - \min_k x_{k,j}}$ [27]. This gives the normalised score of the best algorithm as 1 and the worst as 0. This method fails to account for nonlinear scaling of performance difficulty [21]. It requires that changes in performance be equally challenging. For instance, going from a score of 0.1 to 0.2 is assumed to be of equal difficulty to going from 0.7 to 0.8. This is not true of tasks with non-linear changes in score as the agent improves over difficulty levels, for example in video game levels like Super Mario [21]. Another issue is that if one algorithm has an outlying low score on a particular task, it could make that task seem easy by ensuring all other algorithms have high scores [21].[42]

Policy percentiles [94] offer a means to address the issues of location, scale, non-linear difficulty scaling and magnitude measurement, whilst offering a measurement that can be additively aggregated over [21]. Given a set of policies Π for task j, the cumulative distribution function $F_{X_{\Pi,j}}$ results from the distribution of performances when a policy is sampled uniformly from Π such that policy $\pi \sim U(\Pi)$ [21, 94]. Performance scores $X_{k,j}$ for an algorithm k can be projected through this function to give a normalised score on $[0,1]$ that is scaled according to how difficult it is to achieve the same performance level using a random policy search (from Π). The normalised score for algorithm k with score $X_{k,j}$ is then $F_{X_{\Pi,j}}(X_{k,j})$.

While the scaling according to difficulty is beneficial, this method falls short. As Π has a large search space, the random policy search is unlikely to achieve high scores meaning that many of our desired measurements will approach 1 [21].

Hence, the normalisation approach used by Jordan et al. [21] adapts policy percentiles to try capture the ability to scale algorithms' scores according to difficulty level. Given a set of algorithms $\mathcal{L}$ which act in some task $j \in \Theta$, each of the algorithms having scores $x_{k,j} \in \mathbb{R}$, then each algorithm will have a performance cumulative distribution function $F_{X_{k,j}}(x)$. These functions encode the relative difficulty of algorithm k achieving certain scores on task j, where for a small change in x, a large change in $F_{X_{k,j}}$ corresponds to a relatively high degree of difficulty for algorithm k to improve over. Jordan et al. [21] note that using another algorithm's performance distribution function to normalise a second algorithm's score can shift the distribution of recorded scores of the second algorithm toward 0 when the second algorithm is worse than the first. Similarly, when the second algorithm achieves a greater range of scores, its normalised scores, using the distribution function of the first algorithm, will shift toward 1. Hence, Jordan et al. [21], adopt the use of a weighted average of each algorithm's distribution function as their normalisation function such that for a set of weights $\omega_j \in \mathbb{R}^{|\mathcal{L}|}$ with $\sum_{k=1}^{|\mathcal{L}|} w_{k,j} = 1$, and $\forall k,\ w_{k,j} \geq 0$ then [21]:

$$g(x,j) = \sum_{k \in \mathcal{L}} w_{k,j} F_{X_{k,j}}(x) \tag{19}$$

Jordan et al. [21] label this choice of normalisation function as the *performance percentile* and claim that this captures the relative performance characteristics of a task j for any one algorithm k relative to the others in $\mathcal{L}$. Given this normalisation function for each algorithm's score on a single task j, Jordan et al. [21] desire some aggregate performance measure, $y_k \in \mathbb{R}$ for any one algorithm k, over all tasks, $j \in \Theta$ such that for some weights $q_j \geq 0$ having $\sum_{j \in \Theta} q_j = 1$ [21]:

$$y_k = \sum_{j \in \Theta} q_j \mathbb{E}\left[g\left(X_{k,j}, j\right)\right] \tag{20}$$

Combining the normalisation function in Equation 19 with the aggregation method in Equation 20 gives [21]:

[42] and similarly for an algorithm with an outlying high score, the task would seem particularly difficult.

$$
\begin{aligned}
y_k &= \sum_{j \in \Theta} q_j \mathbb{E}\left[\sum_{n \in \mathcal{L}} w_{n,j} F_{X_{n,j}}(X_{k,j})\right] \\
&= \sum_{j \in \Theta} \sum_{n \in \mathcal{L}} q^*_{n,j} \mathbb{E}\left[F_{X_{n,j}}(X_{k,j})\right]
\end{aligned} \tag{21}
$$

Here, $q^*_{k,j}$ combines weights q over the tasks and weights ω over the algorithms for each task. Jordan et al. [21] desire weights q to prevent overfitting and the exploitation of an algorithm of a particular task to the benefit of its performance. The authors' selection of weights $q^*_{k,j}$ is based on the equilibrium solution of a zero-sum two-player game where the first player tries to select the optimal algorithm $k \in \mathcal{L}$ to maximise aggregate performance defined by the payoff function $\mathbb{E}\left[F_{X_{n,j}}(X_{k,j})\right]$. The second player tries to select the task $j \in \Theta$ and reference algorithm $n \in \mathcal{L}$ to minimise the first player's score. The equilibrium solution therefore selects weights $q^*_{k,j}$ that balance the normalisation function to balance each algorithm's strengths against one another. To find the solution, the authors use the α-rank technique [95] which also allows for the efficient calculation of confidence intervals [21, 96].

Unfortunately, while the evaluation scheme proposed by Jordan et al. [21] offers a coherent guide, and while it is consistent with most desired properties for evaluation, it does not meet our requirement for a computationally tractable evaluation scheme. In Jordan et al. [21], algorithms are evaluated on each task for 10000 runs to attempt to achieve statistical significance. Furthermore they recommend a minimum of 1000 runs per algorithm per task which their procedure relies on. While this may work for computationally light tasks with algorithms that do not require hyperparameter tuning, it does not work well otherwise.[43]

Turning, once again, to the work of Agarwal et al. [23] which focuses on computational tractability within the *few run* regime, we note that the authors propose the use of *min-max* normalisation similarly to Bellemare et al. [27]. As noted above, this method allows algorithm scores the possibility to cluster in different regions of $[0, 1]$ depending on the distribution of scores, maxima and minima. We also noted that this method fails to account for non-linear scaling of performance difficulty with scores, and therefore would require changes in performance scores to be equally challenging. This method, therefore, yields incomparable normalised scores between tasks.

Having considered the widely used normalisation techniques, much room is left for the development of a coherent and computationally tractable normalisation method that offers compatibly additive scores between tasks and is consistent with all the desired properties mentioned. Thus, we have the identified the first answer to our first research question: The problem of finding such a normalisation function is a particular problem with empirical RL algorithm performance evaluations that we will have to address. Additionally, the problem of providing a non-biased weighting when aggregating across tasks is in need of attention. We address these later using multi-criteria decision analysis (MCDA).

Another alternative to the above is, however, proposed by Whiteson et al. [12]. They propose counting how many times algorithm k beats algorithm l in a series of trials and estimating $Pr(X_k \geq X_l) > 0.5$ using the *signtest* [97]. This method ensures a fair pairwise comparison of algorithms across tasks. However, it neglects absolute magnitudes of algorithm scores [21].[44] Agarwal et al. [23] also suggest using this metric, referring to it as the *probability of improvement*. Agarwal et al. [23] calculate the probability of improvement using the Mann-Whitney U-statistic [98]. This is calculated as:

[43] However, it is noted that the distribution of algorithm scores, in Jordan et al. [21], after normalisation and aggregation does reveal sensitivity to hyperparameter tuning due to their random hyperparameter selection procedure [23].

[44] For instance if algorithm k repeatedly scored 1 and algorithm l repeatedly scored 0.99, this would yield a normalised score of 1 regardless of the difference being small [21].

$$P\left(X_k > X_l\right) = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} S\left(x_{k,i}, x_{l,j}\right) \quad \text{where} \quad S(x_k, x_l) = \begin{cases} 1, & \text{if } x_l < x_k \\ \frac{1}{2}, & \text{if } x_l = x_k \\ 0, & \text{if } x_l > x_k \end{cases} \tag{22}$$

Agarwal et al. [23] propose the display of the probability of improvement metric alongside their proposed algorithm performance metric, using stratified aggregation over tasks. To address the problem of data aggregation over runs and tasks, Agarwal et al. [23] turn to the interquartile mean (IQM). This offers a refutation of the use of means and medians in the literature. Where means are sensitive to outliers, the IQM is robust, and where medians require large numbers of runs for statistical claims of improvement, the IQM is statistically efficient [23].[45] The interquartile mean is robust to outliers since it calculates the mean of data between the first and third quartiles, while excluding the first and fourth quartile. If data is ordered as $x_1, ..., x_n$ then the IQM can be calculated as:

$$\bar{x}_{\text{IQM}} = \frac{2}{n} \sum_{i=\frac{n}{4}+1}^{\frac{3n}{4}} x_i \tag{23}$$

The data considered for this calculation includes the $N \times M$ data points, collected from N runs on M tasks. The IQM therefore uses $(N \times M)/2$ data points. The IQM is often referred to as the 25% trimmed mean as it is calculated by trimming the top and bottom 25%s of the data. It can therefore be seen as interpolating between the mean and median which are a 0% and 50% trimmed means respectively [23]. The IQM is, therefore, a better indicator of overall algorithm performance than the sample median as it uses 50% of the data where the sample median depends only on at most two data points as well as the ordering of data. For instance, if a dataset of scores has many zeroes in it, the median is not effected and is therefore a poor indicator of overall performance [23]. It is for this reason that the IQM requires smaller sample sizes than the median. Additionally, the IQM is a less statistically biased estimate than the median and has narrower confidence intervals [23].

Reporting the IQM, thus poses as a robust and efficient aggregator to effectively summarise our algorithm performance data across only a few tasks and runs. Alongside a careful selection of tasks and the use of stratified bootstrap confidence intervals a clear view of an effective performance evaluation procedure is emerging from the work of Agarwal et al. [23]. However, the problems of computational tractability and finding normalised scores that do not have problems with non-linear scaling, location or scale remain. While these problems are addressed in the section on multi-criteria decision analysis (section 2.4), the next subsection briefly discusses the effective reporting of results and how variance in reported algorithm performances can result from variance in evaluation procedure.

2.2.4 Result Reporting, Variance Between Papers and Other Metrics

One important concern in algorithm performance evaluation is the high variance in reported algorithm performances when comparing between research papers [17, 21]. Importantly, this is the case when comparing evaluations between papers of the same algorithm acting in the same environment. As noted in Engstrom et al. [22] and Andrychowicz et al. [100], implementation differences and various code-level optimisations of the same algorithm can account for some of the variance of that algorithm's performance between papers. This adds confounding to the comparison of algorithm performances. Furthermore, the reporting of these implementation details and code optimisations or the conducting of ablation studies is a rarity [1]. The high variance between papers and presence of sources of confounding means that the scientific rigour of the

[45] Agarwal et al. [23] show, in a case study on the Atari 100k benchmark [99], that even with $N = 50$ runs, there is substantial uncertainty in the median and that for a sample with few runs the median is highly variable.

reported algorithm performance, and the level of improvements over other algorithms, can be called into question.

Engstrom et al. [22] find that much of the reported performance improvement for the proximal policy optimisation (PPO) [37] algorithm, a successor to the trust region policy optimisation algorithm (TRPO) [101], results from implementation differences and not from the novel clipping function. Engstrom et al. [22], therefore, propose that authors use ablation studies and act with rigour in analysing the effects of individual implementation details on algorithm performance.

It is common in machine learning to emphasise *wins* over a careful scientific analysis of which particular algorithmic component might cause performance improvement [21, 24, 3]. This distinction is referred to as the emphasis of *competitive testing*, where algorithms compete to achieve the best performance level on various benchmark tasks [21, 24]. This is preferred over *scientific testing* which would involve the careful investigation of hypotheses into the causes of algorithm performance gains, via experimentation.

In this dissertation we are focused on the evaluation procedure in order to consistently and scientifically measure deep RL algorithm performances in a computationally tractable way. While we do not elaborate on how best to answer hypotheses on the causes of performance improvement, the establishment of a scientifically consistent performance measurement procedure may form the backbone of any such analysis.

While differences in implementations are one cause of variance in reported algorithm performance, another cause is the differences in evaluation procedures between papers [17, 23]. Results produced under different evaluation procedures are generally incomparable [23]. It is, therefore, important to emphasise a standardised performance evaluation procedure [1]. This involves imposing a level of consistency among research papers, for instance on the use and reporting of certain metrics and confidence intervals. The IQM and stratified bootstrap confidence intervals recommended by Agarwal et al. [23] offer a potential example of this.

Throughout this section we have referred to recommendations made in the literature which could also be important to maintain consistency with. For instance, the tuning of hyperparameters, the duration of training, the frequency of evaluation intervals, the use of the final metric, the use of evaluation episodes, the number of evaluation episodes and the normalisation method should all be standardised and reported. Henderson et al. [17] investigated the effects of hyperparameter tuning, environmental stochasticity, different random seeds and algorithm implementation details on algorithm performance. They found that each of these features, including the network architectures and the activation functions used, can lead to significant differences in reported algorithm performances. This demonstrated deep RL's performance sensitivity to such details. Therefore, it is also vital to report hyperparameters and publish code [17, 3]. Specifically, Henderson et al. [17] recommend the reporting of all experimental details as well as evaluation procedures.

While only advocating for a few trials to maintain computational tractability of our evaluation procedure [23], we note that it is still vital to have repeated samples for each algorithm on each task so as to maintain a large enough sample to effectively compute meaningful statistics [17, 18]. We also note that it is not uncommon for the cherry-picking of the top N runs for result reporting [1, 21]. We recommend against this biased and non-random sub-sampling of results.

To enhance one's readers view of reported algorithm performance, we recommend the reporting of training curves [19, 20, 23]. In referring to these as sample efficiency curves [23], we note that these can depict how efficiently algorithms can train to achieve a certain level of performance. These curves can also be used for comparison at different timesteps [19]. Additionally, Agarwal et al. [23] suggests using the optimality gap. This provides a measure of how distant an algorithm's performance is away from some optimal performance γ, $\frac{1}{NM}\sum_{j=1}^{M}\sum_{n=1}^{N}(\gamma - x_{j,n})$. This can be plotted for varying levels of γ. Other metrics like the probability of improvement are also recommended [12, 23].[46].

[46] as referred to in subsection 2.2.3

Adapting the probability of improvement by setting one set of scores to human scores yields the *superhuman probability* which gives the number of runs above human performance [23].[47] Additionally, Agarwal et al. [23] mention the *difficulty progress* to be the mean scores of some bottom percentage of runs to see how the algorithms perform on harder tasks. Finally, rank distribution plots provide a visualisation of the probability that a given algorithm is assigned rank $k, k \in \{1, ..., |\mathcal{L}|\}$ for a given metric [23].[48] These are estimated using bootstrapping. Importantly, each metric only reveals a certain aspect of algorithm performance. This is illustrated by algorithm rankings differing depending on the metric used [23]. A more holistic view of the data set is offered by *performance profiles* [23, 112].

As performance scores between tasks may vary, be heavy tailed, multimodal and have outliers, and as point estimates provide an incomplete view of performance, performance profiles are proposed by Agarwal et al. [23]. These provide a plot of the empirical tail distribution function, $\hat{F}(\tau; x_{1:N}) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}[x_n > \tau]$ which shows the proportion of runs above a certain score, τ. This is adapted in Agarwal et al. [23] to become the *run-score distribution* which shows the proportion of runs above a normalised score across all tasks:

$$\hat{F}_X(\tau) = \hat{F}(\tau; x_{1:M,1:N}) = \frac{1}{M}\sum_{j=1}^{M}\frac{1}{N}\sum_{n=1}^{N}\mathbb{I}[x_{j,n} > \tau] \tag{24}$$

This is plotted using stratified bootstrap confidence bands [23]. These allow for the visualisation of the full data set of algorithm scores, $x^k_{1:M,1:N}$ for all algorithms $k \in \mathcal{L}$, tasks $j \in \Theta$ and runs $n \in \{1, ..., N_j\}$. These provide a view of performance in one glance as opposed to the overwhelming view of tables of aggregate metrics [23]. The authors also note that performance profiles are robust to outlier runs and to small changes in algorithm performances across tasks. For instance, an outlier score can change the performance profile plot distribution by at most $\frac{1}{MN}$. Furthermore, the distribution is an unbiased estimator of the true distribution [23].[49] Both the optimality gap and IQM can be read as areas off the performance profile plot. An example of performance profiles can be seen in Figure 6, from Agarwal et al. [23]. An algorithm is said to outperform another algorithm if its curve stochastically dominates the other [23, 113, 114].[50]

With all these plots, metrics and evaluation procedures mentioned in the above subsections we have a better view of current criticisms and recommendations for evaluation in deep RL. The recommendations here will aid in our motion toward a better guideline for performance evaluations. We are, however, left with the two open questions: how to normalise data for coherent aggregation and how to weight the importance of each task for this aggregation. The identification of these two problems serves as a response to our first research question: *Are there specific problems with deep RL empirical algorithm performance evaluations and why are these an issue?*

Next, it is useful to turn to examine the state of a recently popular subset of the deep RL literature: multi-agent reinforcement learning (MARL). As the works discussed in this section have only analysed the evaluation procedures commonly used in single-agent RL (SARL) research publications, and as the MARL

[47] The ability of RL algorithms to achieve strong performances in grand-challenge problems or against human experts is a popular means of framing algorithmic evaluation. Early successes of RL involved measuring performance on the game of backgammon [102]. More recent developments saw the AlphaGo series [28, 103, 104] achieve success over the world champion of the game of Go. This paradigm of setting challenges for agents to overcome has been extended to MARL too. StarCraft II [105, 42], poker [106, 107, 108] and Dota 2 [109] provided challenges where MARL algorithms were able to achieve human or superhuman level performances. Capture-the-flag [110] and hide-and-seek games [111] have also supplied the MARL field with challenging benchmark environments in which human comparisons can be drawn [6].

[48] Here $\mathcal{L}$ is the set of algorithms.

[49] Agarwal et al. [23] contrast their performance profiles with *average-score distributions* [27]. The average score distribution is the performance profile of $\bar{x}$ such that $\hat{F}_{\bar{X}}(\tau) = \hat{F}(\tau; \bar{x}_{1:M}) = \frac{1}{M}\sum_{j=1}^{M}\mathbb{I}[x_j > \tau]$. The average score distribution is a biased estimator, is a step function in $\frac{1}{M}$ (as opposed to $\frac{1}{MN}$) and has higher variance $\sigma^2_{\bar{X}} = \frac{1}{M^2}\sum_{j=1}^{M} F_{\bar{X}_j}(\tau)\left(1 - F_{\bar{X}_j}(\tau)\right)$ (as opposed to $\sigma^2_X = \frac{1}{M^2 N}\sum_{j=1}^{M} F_j(\tau)(1 - F_j(\tau))$) [23].

[50] Stochastic dominance of a variable X over another Y happens if $\forall\tau, Pr(X > \tau) \geq Pr(Y > \tau)$ and for some $\tau, Pr(X > \tau) > Pr(Y > \tau)$ [23].

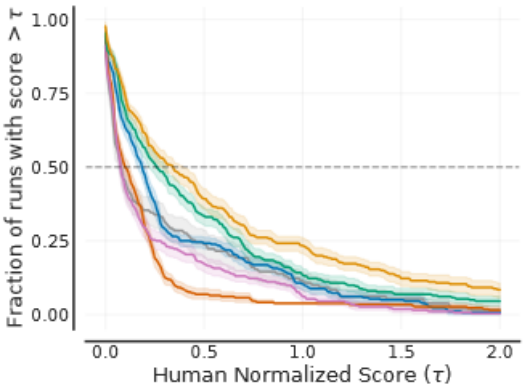

Figure 6: An example of a performance profile for six algorithms on the Atari 100k environment [99]. This appears in Agarwal et al. [23]. The yellow algorithm, SPR [115], appears to outperform the others.

research field has only a few publications which mention such issues, we find it valuable to use MARL for a case study into the level of rigour of recent RL publications.

The next subsection examines the current state of empirical algorithm performance evaluation procedures in deep cooperative MARL.[51] This will help answer question (ii) of the introduction to this dissertation: *What is the level of rigour in deep cooperative MARL empirical algorithm performance evaluations?* The questions of normalisation and weighting will be addressed thereafter when introducing multi-criteria decision analysis.

2.3 Evaluation in Cooperative MARL

Cooperative MARL can be used to address the RL problem at scale [4]. As mentioned above, the cooperative MARL setting has all agents sharing a reward function and therefore sharing objectives [47]. It is for this reason that the evaluation of the performance of cooperative MARL algorithms is equivalent to the evaluation of SARL algorithm performance. For example, in the case that a task's control is centralised, that task then reduces to an MDP [116]. Hence, this particular task may be viewed as a task that is solvable using SARL. In this case, cooperative MARL only seeks to provide a performance benefit. Cooperative MARL therefore provides a simple framework for extending the capabilities of SARL. This is why much contemporary research has focused on cooperative MARL specifically[52] and this, in turn, is why the focus of this section is on cooperative MARL only.

Despite the simplicity of cooperative MARL performance evaluation (in terms being analogous to SARL algorithm performance evaluation) the ability of the MARL algorithm to converge to a global optimum (or equilibrium) is still difficult to evaluate [47, 116]. In contrast to SARL, coordination between agents is required to reach such a global optimum. This may be achieved through means of communication [60, 72, 117]. Much contemporary research has gone into the design of effective cooperative MARL communication algorithms for example with the *CommNet* [72], *IC3NET* [118], *DGN* [119], *TARMAC* [120], *RIAL* and *DIAL* [60], *MagNet* [121], *GA-Comm* [122], *Agent-Entity Graph* [123], *VBC* [124], *TMC* [125] and *MAGIC* [126] algorithms. Game theoretic analyses such as that used in Pretorius et al. [44] may be useful in evaluating certain design choices pertaining to network topologies and communication protocols. In contrast, Yang and Wang [6] notes that many algorithms, such as those that use Q-function decomposition, lack theoretical backing and rely instead on empirical demonstration. While such game theoretic evaluations are useful, they do not form the focus of this section and instead we turn to *empirical* algorithm performance evaluations such as those discussed for SARL, in section 2.2 above.

Importantly, the lessons learnt from the above work, in the field of deep SARL empirical algorithm per-

[51] Cooperative MARL is specifically studied since its evaluations are equivalent to those in SARL.
[52] as opposed to focusing on competitive and mixed games.

formance evaluation, pertain to cooperative MARL. As mentioned, this is due to the agents in cooperative MARL having a shared reward function, the evaluation of their empirical performance of their shared governing algorithm is equivalent to the evaluation of a SARL algorithm. Furthermore, cooperative MARL suffers from many of the same issues that plague evaluation in SARL [1, 63, 75]. MARL algorithm performances vary due to sensitivity to hyperparameter tuning, implementation details and different random seeds. Moreover, inconsistencies in evaluation procedures are rife [1, 88]. The lessons learnt from the SARL evaluation literature will therefore be useful in, and transferable to, cooperative MARL. We will use these lessons in section 3 to construct a protocol for RL evaluation.[53]

In this section we survey the evaluation methods used in recent cooperative MARL literature. We do this via the study of a dataset which documents the cooperative MARL evaluation methods found in 75 published papers [1]. The dataset will reveal that similar issues as those plaguing SARL also affect cooperative MARL. The dataset also reveals that there is an overall need to improve standardisation and statistical rigour in the field. Some of these issues are observed and addressed in some recent MARL literature [63, 88, 75].

Recent works in cooperative MARL have attempted to provide the field with benchmark environments and benchmark algorithms for evaluation [81]. This can be seen with PyMARL [81], which includes several implementations of popular cooperative MARL algorithms, and the *Starcraft Multi-Agent Challenge* (SMAC) [81] suite of benchmark tasks which compares to the SARL Arcade Learning Environment (ALE) [27] and MuJoCo [127]. Similarly Leibo et al. [80] provide *Melting Pot*. This is a configurable task suite that provides 80 test scenarios to test multiple aspects of MARL agents' behaviours.

Ablation studies are demonstrated to be of value when assessing code optimisations and implementation tricks. In Yu et al. [63], the authors conduct ablation studies of the implementation differences specific to the PPO algorithm. They also reveal *death-masking*, a SMAC-specific environment optimisation, to be of benefit to PPO implementations. Additionally, Hu et al. [75] investigate implementation tricks and code optimisations associated with the popular *QMIX* algorithm. They demonstrate significant variance in the performance of QMIX for different implementations of the same algorithm. The evaluations conducted in these papers demonstrate the benefit of asking *why* it is that an algorithm's performance differs from previous implementations.

Similarly, Papoudakis et al. [88] investigate why it is that certain types of algorithms perform better in certain types of tasks. In doing so they demonstrate the benefit of using multiple environments to evaluate an algorithm's performance. They investigate the centralised training, decentralised execution (CTDE) and the independent learning paradigms on multiple environments and reveal that the previously popular value-decomposition algorithms, which perform well on the SMAC environment, struggle to learn in the sparse reward *RWARE* [128] environment as compared to independent learning algorithms. Papoudakis et al. [89] therefore address the issue of variance in performance over multiple environments, stating that algorithm performance depends strongly on task properties and that no one algorithm learns efficiently across all tasks.[54]

Hence, where the above works focus on performing new performance assessments, performing implementation optimisations or providing new benchmarks, our work focuses on addressing the overall rigour by conducting a case study on the level of evaluation rigour in cooperative MARL research. The view in this dissertation, as echoed by Sculley et al. [2] and Forde and Paganini [3], is that standardisation of evaluation methodology, better statistical rigour and standardised choices of benchmarking will provide a clearer depiction of the performance of any one algorithm and will yield a better framework for consistent and valid algorithm comparison.

In contrast to the last subsection, this subsection addresses the dearth of literature in MARL performance evaluation and instead performs an exploratory data analysis of the aforementioned dataset by Gorsane et al. [1], comprising published MARL research papers. This analysis contributes toward addressing the

[53] This will generally apply to both SARL and cooperative MARL.
[54] This is in line with the *no free lunch principle* [129].

dearth as it works toward diagnosing the level of consistency and rigour in cooperative MARL evaluation. The lessons learnt here will later be used when constructing our guideline for evaluation.

The dataset[55] contains the deep cooperative MARL algorithm performance evaluation procedures which were used in 75 papers, published between the years of 2016 and 2022. The dataset comprises all recent popular deep MARL algorithms as well as influential papers [1]. It therefore documents the evaluation procedures of one particular (cooperative MARL) subset of recent deep RL publications. This exploratory data analysis thus constitutes a case study into the level of rigour of a recent subset of RL literature. Table 2 documents the research papers from which the cooperative MARL evaluation data were recorded.

[55] which can be found in the supplementary material of Gorsane et al. [1], at https://marl-dataset.notion.site/marl-dataset/MARL-dataset-d632230523a74f2793630504ab4542e5.

Table 2: Below are the deep cooperative MARL research publications from which the dataset by Gorsane et al. [1] collected their data on algorithm performance evaluation procedures. The data was recorded from 75 published research papers. The data collection methodology is found in Gorsane et al. [1].

Title	Authors
Learning Multiagent Communication with Backpropagation	Sukhbaatar et al. [72]
Learning to Communication in Deep Multi-Agent Reinforcement Learning	Foerster et al. [60]
Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments	Lowe et al. [58]
Efficient Communication in Multi-Agent Reinforcement Learning via Variance Based Control	Zhang et al. [124]
LIIR: Learning Individual Intrinsic Reward in Multi-Agent Reinforcement Learning	Du et al. [130]
MAVEN: Multi-Agent Variational Exploration	Mahajan et al. [131]
Multi-Agent Common Knowledge Reinforcement Learning	Schroeder de Witt et al. [132]
A Structured Prediction Approach for Generalization in Cooperative Multi-Agent Reinforcement Learning	Carion et al. [133]
Succinct and Robust Multi-Agent Communication With Temporal Message Control	Zhang et al. [125]
Learning Multi-Agent Coordination for Enhancing Target Coverage in Directional Sensor Networks	Xu et al. [134]
Promoting Coordination through Policy Regularization in Multi-Agent Deep Reinforcement Learning	Roy et al. [135]
Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning	Christianos et al. [128]
Learning Individually Inferred Communication for Multi-Agent Cooperation	Ding et al. [136]
Learning Implicit Credit Assignment for Cooperative Multi-Agent Reinforcement Learning	Zhou et al. [137]
Variational Automatic Curriculum Learning for Sparse-Reward Cooperative Multi-Agent Problems	Chen et al. [138]
Pessimism Meets Invariance: Provably Efficient Offline Mean-Field Multi-Agent RL	Chen et al. [139]
Towards Understanding Cooperative Multi-Agent Q-Learning with Value Factorization	Wang et al. [140]
Investigation of Independent Reinforcement Learning Algorithms in Multi-Agent Environments	Lee et al. [141]
Celebrating Diversity in Shared Multi-Agent Reinforcement Learning	Chenghao et al. [142]
Weighted QMIX: Expanding Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning	Rashid et al. [143]
Episodic Multi-agent Reinforcement Learning with Curiosity-driven Exploration	Zheng et al. [144]
Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms in Cooperative Tasks	Papoudakis et al. [88]
Regularized Softmax Deep Multi-Agent Q-Learning	Pan et al. [145]
Settling the Variance of Multi-Agent Policy Gradients	Kuba et al. [146]
FACMAC: Factored Multi-Agent Centralised Policy Gradients	Peng et al. [147]
Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning	Foerster et al. [148]
Actor-Attention-Critic for Multi-Agent Reinforcement Learning	Iqbal and Sha [149]
Deep Decentralized Multi-task Multi-Agent Reinforcement Learning under Partial Observability	Omidshafiei et al. [150]
QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning	Rashid et al. [61]
Social Influence as Intrinsic Motivation for Multi-Agent Deep Reinforcement Learning	Jaques et al. [151]
TarMAC: Targeted Multi-Agent Communication	Das et al. [120]
QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning	Son et al. [70]
Deep Coordination Graphs	Boehmer et al. [152]
DFAC Framework: Factorizing the Value Function via Quantile Mixture for Multi-Agent Distributional Q-Learning	Sun et al. [153]
Scaling Multi-Agent Reinforcement Learning with Selective Parameter Sharing	Christianos et al. [154]
The Emergence of Individuality	Jiang and Lu [155]
Cooperative Exploration for Multi-Agent Deep Reinforcement Learning	Liu et al. [156]
MMD-MIX: Value Function Factorisation with Maximum Mean Discrepancy for Cooperative Multi-Agent Reinforcement Learning	Xu et al. [157]
Domain-Aware Multiagent Reinforcement Learning in Navigation	Saeed et al. [91]
Local Advantage Actor-Critic for Robust Multi-Agent Deep Reinforcement Learning	Xiao et al. [158]
MAGNet: Multi-agent Graph Network for Deep Multi-agent Reinforcement Learning	Malysheva et al. [121]
PIC: Permutation Invariant Critic for Multi-Agent Deep Reinforcement Learning	Liu et al. [159]
Centralizing State-Values in Dueling Networks for Multi-Robot Reinforcement Learning Mapless Navigation	Marchesini and Farinelli [160]
Learning when to Communicate at Scale in Multiagent Cooperative and Competitive Tasks	Singh et al. [118]
Influence-Based Multi-Agent Exploration	Wang et al. [161]
Action Semantics Network: Considering the Effects of Actions in Multiagent Systems	Wang et al. [162]
Learning Nearly Decomposable Value Functions Via Communication Minimization	Wang et al. [163]
Evolutionary Population Curriculum for Scaling Multi-Agent Reinforcement Learning	Long* et al. [164]
Simplified Action Decoder for Deep Multi-Agent Reinforcement Learning	Hu and Foerster [165]
RODE: Learning Roles to Decompose Multi-Agent Tasks	Wang et al. [166]
QPLEX: Duplex Dueling Multi-Agent Q-Learning	Wang et al. [167]
LIGS: Learnable Intrinsic-Reward Generation Selection for Multi-Agent Learning	Mguni et al. [168]
ToM2C: Target-oriented Multi-agent Communication and Cooperation with Theory of Mind	Wang et al. [169]
Trust Region Policy Optimisation in Multi-Agent Reinforcement Learning	Kuba et al. [170]
Reinforcement Learning for Location-Aware Warehouse Scheduling	Stavroulakis and Sengupta [171]
Multi-agent Transfer Learning in Reinforcement Learning-based Ride-sharing Systems	Castagna and Dusparic [172]
MultiAgent Soft-Q Learning	Wei et al. [173]
Counterfactual Multi-Agent Policy Gradients	Foerster et al. [62]
Multi-Agent Game Abstraction via Graph Attention Neural Network	Liu et al. [122]
Shapley Q-value: A Local Reward Approach to Solve Global Reward Games	Wang et al. [174]
SMIX(λ): Enhancing Centralized Value Functions for Cooperative Multi-Agent Reinforcement Learning	Wen et al. [175]
QVMix and QVMix-Max: Extending the Deep Quality-Value Family of Algorithms to Cooperative Multi-Agent Reinforcement Learning	Leroy et al. [176]
Value-Decomposition Multi-Agent Actor-Critics	Su et al. [73]
Evaluating Generalization and Transfer Capacity of Multi-Agent Reinforcement Learning Across Variable Number of Agents	Guresti and Ure [177]
Multi-Agent Incentive Communication via Decentralized Teammate Modeling	Yuan et al. [178]
A Deeper Understanding of State-Based Critics in Multi-Agent Reinforcement Learning	Lyu et al. [179]
Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward	Sunehag et al. [68]
Modelling the Dynamic Joint Policy of Teammates with Attention Multi-agent DDPG	Mao et al. [180]
The StarCraft Multi-Agent Challenge	Samvelyan et al. [81]
Feudal Multi-Agent Hierarchies for Cooperative Reinforcement Learning	Ma and Wu [181]
Learning Transferable Cooperative Behavior in Multi-Agent Teams	Agarwal et al. [123]
Comparative Evaluation of Cooperative Multi-Agent Deep Reinforcement Learning Algorithms	Papoudakis et al. [89]
Deep Implicit Coordination Graphs for Multi-agent Reinforcement Learning	Li et al. [182]
Off-Policy Correction For Multi-Agent Reinforcement Learning	Zawalski et al. [183]
Local Advantage Networks for Cooperative Multi-Agent Reinforcement Learning	Avalos et al. [184]

The dataset was collected as part of a group effort in this dissertation's accompanying publication, Gorsane et al. [1].[56] It was collected to primarily answer question (ii) from the introduction of this dissertation: *What is the level of rigour in deep cooperative MARL empirical algorithm performance evaluation?* Here, we provided an exploratory hypothesis, that the gathered data will demonstrate a poor level of rigour in the empirical performance evaluations of deep cooperative MARL algorithms. This exploratory hypothesis stems from our literature review on SARL evaluation (section 2.2), the equivalent characteristics between deep SARL and deep cooperative MARL algorithm performance evaluation procedures and the several works (briefly discussed above) examining inconsistencies in benchmarking deep cooperative MARL algorithms [63, 75, 88]. We answer the above question focusing on some significant plots from the data set.[57]

Figure 7 shows distributions of reported algorithm performances from the deep cooperative MARL literature. Three algorithms, Q-Mix [61], VDN [68] and IQL [55], are studied within two classes of environments, SMAC [81] and MPE [58]. The plots demonstrate significant spreads of reported algorithm performances between the different pieces of literature. Significant differences in reported algorithm performances appear consistently across all three algorithms and across most of the tasks plotted. This immediately alludes to inconsistencies in the cooperative MARL literature, which draws similarity to the variance in reported performances found in the SARL literature [17, 21]. Even though many of the provided data points are measurements of averages across multiple algorithm runs, the reported performances still differ between papers. Thus we ask: From where do these inconsistencies arise?

As expressed by Yu et al. [63] and Hu et al. [75], the inconsistencies in MARL algorithm performances between papers may arise from various implementation differences within the same algorithm. To ascertain the source of this type of inconsistency, one can follow the practice in Hu et al. [75] and make use of ablation studies, which are recommended scientific practice [1]. The publishing of open-source code and implementation details is also important [1, 17].

While one source of inconsistency arises from implementation differences, others may arise from the version of published tasks or environments as well as the training time. Gorsane et al. [1] find that even when controlling for these factors, significant reported algorithm performance differences still arise. Differences in evaluation methodology between papers, thus, poses a significant area of investigation for the differences between reported performances.

We will refer back to the data analysis of inconsistent evaluation methodologies shortly. Here, we note that these inconsistencies, along with those referred to above, lead to inconclusive results in terms of the ability to rank algorithms. While Papoudakis et al. [88] find that no one algorithm learns efficiently across all tasks, our data analysis finds that any one algorithm's performance may vary across papers even for one task. Hence our results suggest that the set of performance rankings, obtained from the competitive testing [21, 24] of a set of algorithms on a single task, are inconclusive. Furthermore the repeated reproduction of such an experiment may produce a different set of rankings each time.

Figure 8 acts to demonstrate the point made by Papoudakis et al. [89]. When comparing a set of algorithm performance rankings on one task, with another set of performance rankings on another task, one may find inconsistencies. This further emphasises the need for evaluating algorithm performances across multiple tasks, as emphasised by Whiteson et al. [12], Jordan et al. [21] and Agarwal et al. [23]. Figure 9 (b) shows that many papers make use of only a few tasks to evaluate algorithms on. This, thereby demonstrates the risk of environment overfitting, as posed by Whiteson et al. [12], Zhang et al. [16] and Zhang et al. [74].

Figure 9 (a) demonstrates that a disproportionate number of papers in the deep cooperative MARL liter-

[56] Details of data collection can be found in Gorsane et al. [1]. I contributed to collecting some of the papers and recording some of the data. More details on the publication can be found in the statement of contribution at the beginning of this dissertation.

[57] Other interesting plots and data analysis can be found in Gorsane et al. [1] and its supplementary material. All plots found in this section of the dissertation can be reproduced using the supplementary notebook called EDA_RL_Evaluation_Literature.ipynb found at https://github.com/marlEvalDissertation/marlEvalDiss. The notebook is written by me and contains code adapted from the data analysis notebook of Gorsane et al. [1].

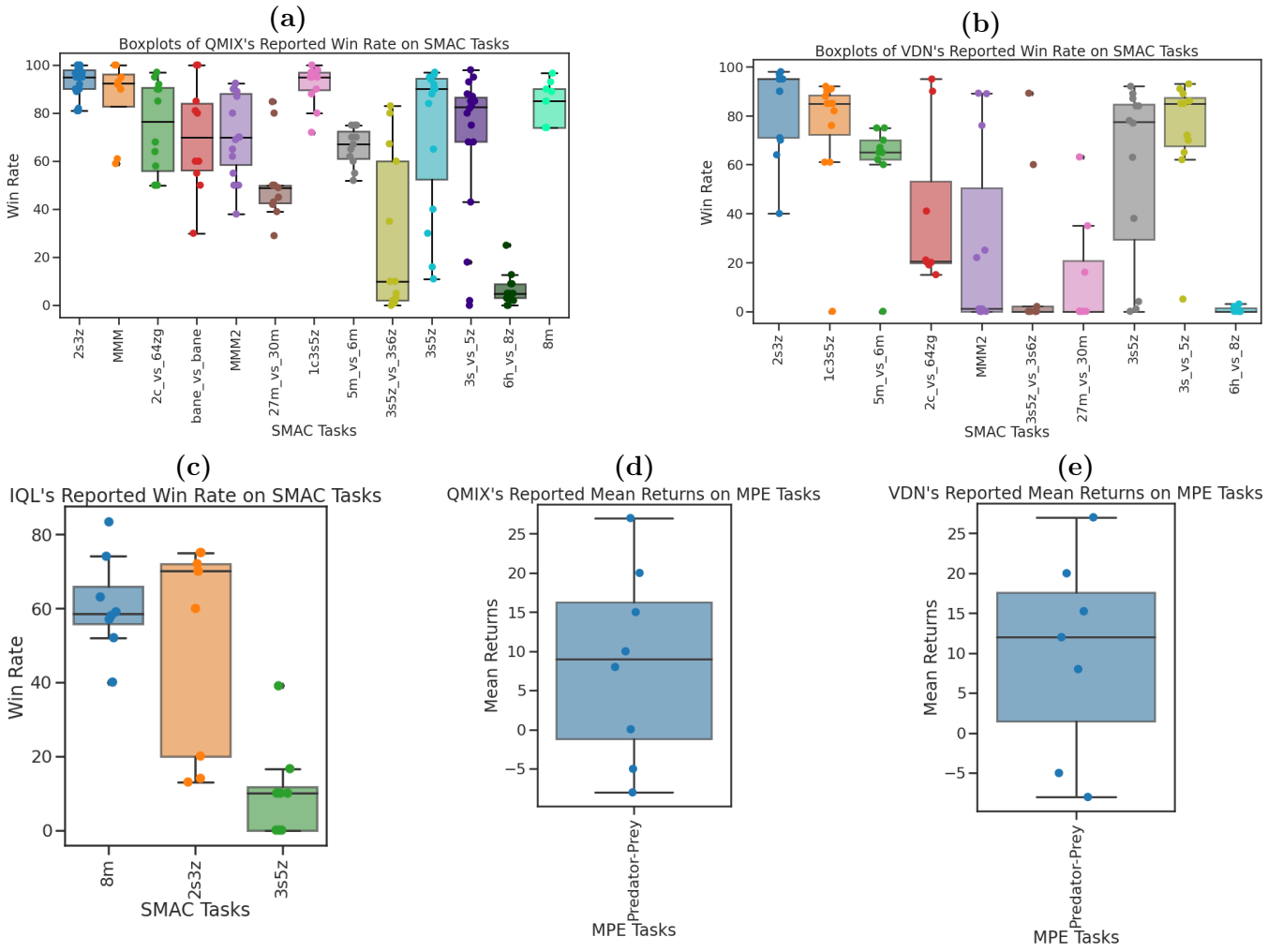

Figure 7: Plots of the reported empirical evaluations of deep cooperative MARL algorithm performances. Algorithm performances are plotted from a subset of deep cooperative MARL literature which was collected in a dataset by Gorsane et al. [1]. The boxplots above show a large spread in reported algorithm performances, indicating inconsistencies. **(a)** Reported win rates of the Q-MIX algorithm [61] for various tasks found in the SMAC [81] environment. **(b)** Reported win rates of the value decomposition networks (VDN) [68] algorithm in SMAC. **(c)** Reported win rates for the independent Q learning (IQL) [1, 55] algorithm in SMAC. **(d)** Reported mean returns for Q-Mix for the predator-prey task in the MPE [58] environment. **(e)** Reported mean returns for VDN in the predator-prey task.

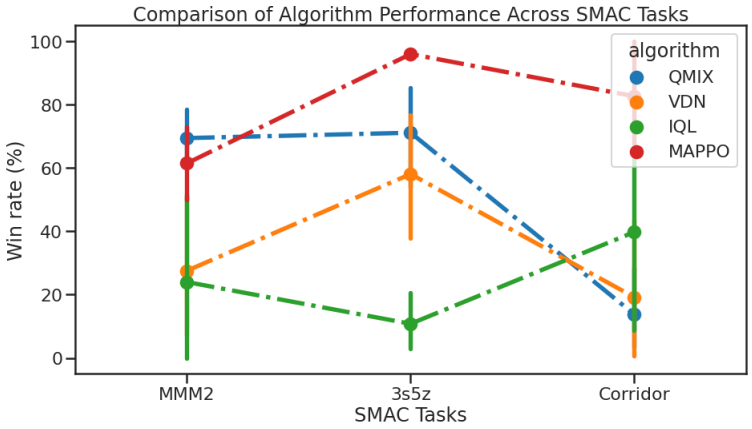

Figure 8: Reported, aggregated deep cooperative MARL empirical algorithm performances on three SMAC [81] tasks. The performance point estimates are aggregated across publications in the cooperative MARL literature which were collected in a dataset by Gorsane et al. [1]. The plot demonstrates that the algorithm ranking is inconsistent between the three tasks.

ature focus on performance evaluations using SMAC [81] or MPE [58]. While both of these offer multiple training tasks for MARL algorithms, their disproportionate overuse may be a sign of environment overfitting [1]. The use of too few tasks, as depicted by Figure 9 (b), is a further indication of environment overfitting. For instance, a researcher can randomly select a subset of tasks where one algorithm performs well over another. However if a different random subset of tasks was selected, the rankings of the algorithms could change. This is illustrated by Figure 8. While environment overfitting offers a further problem in cooperative MARL evaluation literature, we now turn to inconsistencies in the evaluation methodologies found in the literature.

As depicted in Figure 10, the choices of uncertainty measures, aggregation methods and number of runs used are inconsistent across the deep cooperative MARL literature. Furthermore, as depicted in Figure 10 **(a)**, it is not uncommon for only point estimates to be reported, without specification of some measure of uncertainty. Moreover, as depicted in Figure 10 **(b)**, it is not uncommon for various aspects of a paper's performance evaluation protocol to go unreported. The lack of reporting of evaluation details mean that the replicability of a paper's results becomes unattainable. Gorsane et al. [1] notes that details such as evaluation frequency and evaluation duration also often go unreported.

While inconsistencies in levels of reporting can stunt effective replicability, inconsistencies in evaluation methodologies lead to the stunting of reproducibility of any experiment. Differences in experimental procedure such as the variable choice of the number of independent training runs to use, as seen in Figure 10 **(c)**, offer another reason for the variance seen in reported algorithm performances (Figure 7).

Hence, having analysed the dataset of MARL evaluations [1], we can provide a diagnosis of the various problems seen in the deep cooperative MARL literature. The literature contains inconsistencies in reported algorithm performances and ranks of algorithms, as seen in Figures 7 and 8. This is true even for evaluation of one algorithm on the same task across papers. These inconsistencies arise from various implementation differences as expressed in Yu et al. [63] and Hu et al. [75]. They arise from inconsistent environment selection and the cherry picking of favourable results from a sub-sample of tasks [1], as well as environment overfitting (Figures 8 and 9). Lastly, the inconsistencies arise from differences in the evaluation protocols used (Figure 10).

The exploratory data analysis conducted in this section provides evidence toward the confirmation of our exploratory hypothesis of a poor level of rigour in the empirical performance evaluations of deep cooperative MARL algorithms. This leads to concern as to the adequacy and validity of any conclusions drawn about

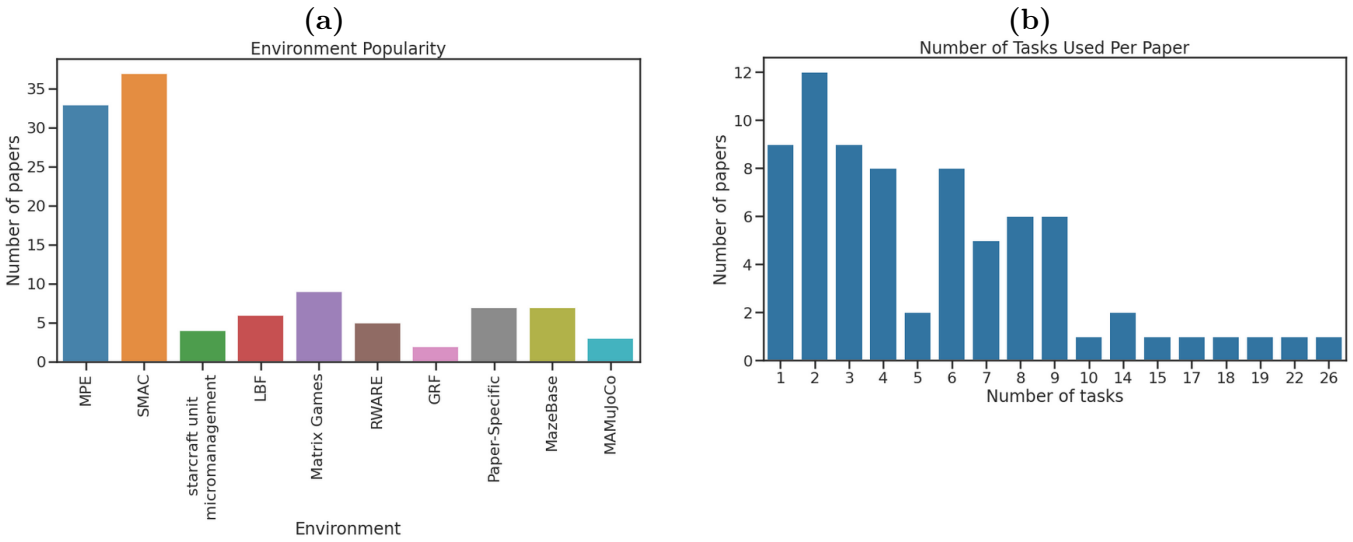

Figure 9: Plots demonstrating the usage of various environments and tasks in the deep cooperative MARL literature, collected in a data set by Gorsane et al. [1]. **(a)** The popularity of various learning environment classes in the deep cooperative MARL literature, measured by the number of papers that make use of each environment class. **(b)** This plot shows how many cooperative MARL learning tasks each paper in the cooperative MARL literature makes use of.

algorithm performances in the RL literature.[58] Moreover, poor standards of uncertainty estimation, the absence of reporting of evaluation methodologies (Figure 10) and the absence of a consistent standard for evaluations justify the need for work to address RL algorithm evaluations. This effort will involve attempting to formulate a guideline for improved performance evaluations.

The lessons learnt from the RL literature will be used in the construction of this guideline. We have discussed lessons on the choices of environment, how to collect data, how to aggregate data and how to take uncertainty measurements. We are left with the problem, however, of how to normalise performance scores so as to not require linear changes in performance scores with changes in task difficulty, and the problem of how to weight tasks so as not to overvalue a certain task in our performance aggregation. The next section begins working toward a guideline by introducing a framework for making a decision on how to choose the best algorithm from a set of algorithms based on its performances on various tasks. The introduction of this framework, called multi-criteria decision analysis, offers a path toward addressing the problem of score normalisation and the problem of weighting tasks for better aggregation.

[58] This is due to cooperative MARL being a recent subset of the RL research field.

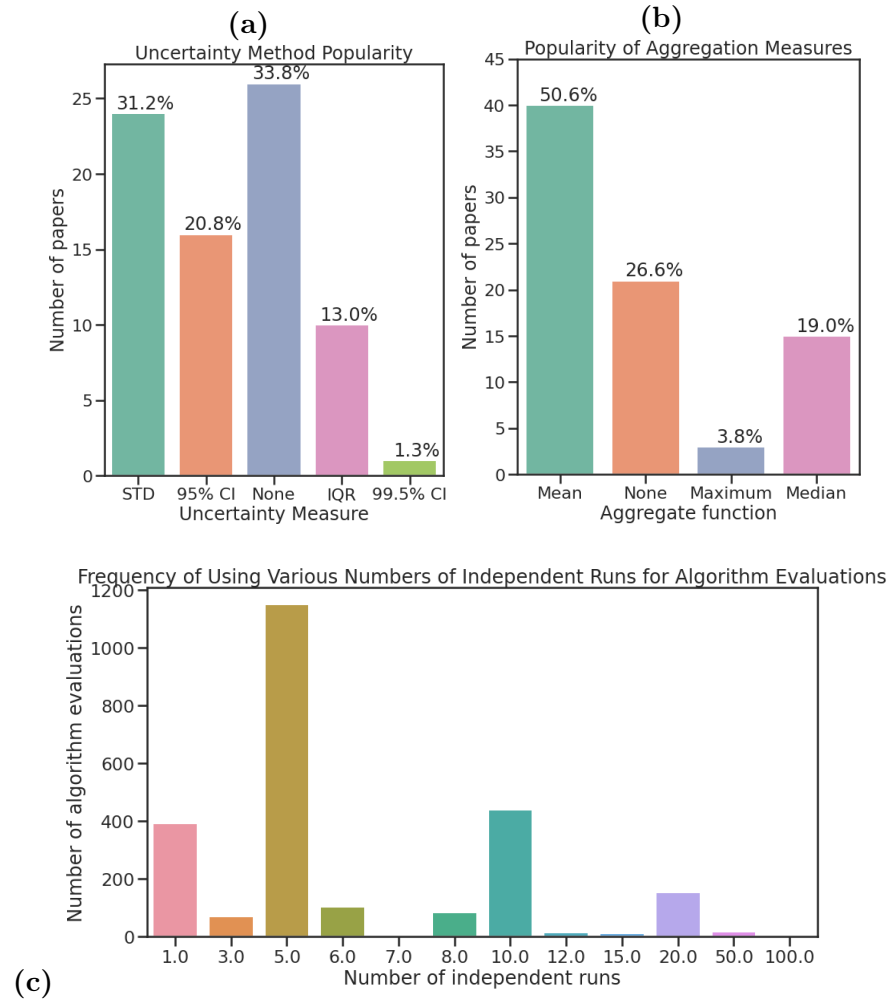

Figure 10: Plots demonstrating the popularity of various aspects of performance evaluation in the deep cooperative MARL literature. These aspects of evaluation were collected in a dataset of MARL literature evaluation, by Gorsane et al. [1]. **(a)** The popularity of uncertainty methods reported in the MARL literature. Here, *none* refers to a lack of uncertainty measure used. *STD* refers to standard deviation. **(b)** The popularity of various aggregation methods used to aggregate across evaluation runs. Here *none* refers to a lack of reported aggregation measure. **(c)** This plot shows the frequency by which various numbers of independent algorithm runs are used as a measure of the number of algorithm performance evaluations that use that many runs to generate performance estimates.

2.4 Multi-Criteria Decision Analysis

One field that addresses the problem of deciding upon an item from a set of alternatives, given multiple criteria for evaluating said items, is the field of multi-criteria decision analysis (MCDA) [25, 26][59]. MCDA provides a coherent and consistent mechanism for performing such a selection. As the selection of an algorithm, from a set of algorithms, based on performances over multiple environments, scenarios, settings, implementations, random seeds and hyperparameters can be seen as a multiple criteria decision problem, then MCDA can be seen as an appropriate solution framework to resolve such a problem. Here, we provide a brief overview of the necessary components of MCDA that we will later use to benefit our guideline for deep RL algorithm performance evaluation.

MCDA involves three broad steps [25, 26]:

1. It involves problem identification and structuring [26]. This primarily involves the explicit identification of criteria for evaluation, and the identification (or development) of alternatives to be compared.
2. MCDA involves model building and preference modelling [25, 26]. This may involve the use of some constructed value functions to ascribe numerical values to each of the alternatives for each criteria [26]. This also involves the aggregation of the assessed values of each alternative, across all criteria [25]. Aggregation often involves the use of normalisation techniques to establish scores of compatible scale across criteria, and the use of weights to measure the importance or relative value of each criteria against one another [26]. This, therefore, addresses the two unsolved problems that we have been left from our review of the RL evaluation literature: normalising the performance scores and weighting the tasks.
3. Lastly, the MCDA process involves the *evaluation* of alternatives with respect to the individual criteria identified, assessment of preferences and development of action plans [26].

MCDA is a sub-field of operations research [26], and has found uses in many fields in business, government and medicine. For instance it has found use in sustainable energy decision-making [185], strategic decision-making [186], natural resource management [187], the selection of medical technologies [188] and in business analytics [189]. Below, we describe the three stages of MCDA in more detail. Defining the problem area forms part of the first stage, problem structuring.

2.4.1 Problem Identification

The first stage in MCDA involves problem identification. MCDA is useful for addressing the following broad classes of situations as identified by Scott [25], Roy and McCord [190] and Belton and Stewart [191]:[60]

- *The choice problem*: This is where a decision will involve the choosing of one out of a set of possible alternatives.
- *The sorting problem*: Here, the decision involves the sorting of a set of alternatives into broad categories.
- *The ranking problem*: This results in a ranking of the alternatives.
- *The description problem*: The aim here is to systematically understand possible actions.

[59] MCDA is also often referred to as multi-criteria decision-making, multi-objective decision-making and multi-attribute decision-making [25].

[60] The broad decision-making areas here, are not, however, mutually exclusive [25].

- *The design problem*: This is to identify new alternatives.
- *The portfolio problem*: This is to select a subset of alternatives from a larger set.

In the case of RL algorithm evaluations we might be seen to be engaging in the choice problem as we are trying to choose one algorithm from a set of possible alternatives. Alternatively, we could be seen to be engaging in ranking the algorithms. Although these are not the only options, having identified a problem type, we can start using the MCDA problem structuring approach.

2.4.2 Problem Structuring

As described above, problem structuring involves defining the problem area that we would like to analyse. Problem structuring may involve the use of the *CAUSE* mnemonic which refers to the identification of *criteria, alternatives, uncertainties, stakeholders and environmental constraints* [26].

Criteria for evaluation can be identified using a tree-like structure where an analyst may, first, identify broad level criteria which can be broken down into sub-criteria. The tree is referred to as a value tree. This process can also be reversed into a bottom-up approach [25, 192]. The analyst will have to balance the trade-off between exhaustiveness and conciseness [25, 193].[61] One must balance computational complexity with having enough criteria to distinguish between alternatives appropriately. One also needs criteria that are measurable and, lastly, have the property of preferential independence [25, 26].

Preferential independence is defined as follows: A criterion J_1 is preferentially independent from another criterion J_2 if the rank ordering of preferences of the second criterion does not change the preference order of the first criterion [25, 26, 194]. Denoting a preference relation by $\succ$ then, preferential independence of J_1 from J_2 has that if the pair $(J_1 = j_1, J_2 = j_2) \succ (J_1 = j_1', J_2 = j_2)$ for all elements $j_1, j_1' \in J_1$ and for some $j_2 \in J_2$ then, $\implies (J_1 = j_1, J_2 = j_2') \succ (J_1 = j_1', J_2 = j_2')$ for all $j_2' \in J_2$. In other words, preferences for specific outcomes of J_1 does not depend on the outcomes of J_2 and it is necessarily possible for an evaluator to evaluate trade-offs between two or more criteria using the assumption that all other criteria have fixed levels of performance (without the need to consider what these fixed levels are) [195]. Moreover, criteria $J_1, ..., J_M$ are mutually preferentially independent if all subsets of $\{J_1 \times ... \times J_M\}$ are preferentially independent of their complement. In the case of this dissertation we aim to choose a set of (preferentially independent and measurable) criteria that correspond to the tasks, $j \in \Theta$, that deep RL algorithms will be used for.

Next, the MCDA problem structuring process requires the evaluation or exploration of **alternatives**. This generally refers to the specification of what alternative decisions the decision-maker can choose between [25]. These can include discrete choices, action plans or strategies. In some circumstances alternatives are clearly defined, while in others, alternatives are discovered via the MCDA process [194]. In the case of this dissertation our set of alternatives are the deep RL algorithms, $k \in \mathcal{L}$.

The consideration of **uncertainties** is also typical of the MCDA process [194]. There are two types of uncertainties to consider. Firstly, internal uncertainties refer to the uncertainty as to the structure of the MCDA model adopted and, therefore, criteria chosen to be included in the model. For instance, in our RL example, this may be uncertainty as to which tasks to assess our algorithms on. This may also involve uncertainty as to what level of detail to include in our assessments, and how many tasks to include. MCDA should allow for the restructuring of the model due to the unresolvable nature of some internal uncertainties [194].

A second kind of internal uncertainty stems from measurement of performance scores from criteria and the elicitation of values of these scores [194]. This refers to imprecision and may include uncertainty

[61] This can be likened to *Occam's Razor*, also known as *the principle of parsimony*.

measurements of quantitative performances. It may also include uncertainty in the trade-offs between performances on different criteria, as well as uncertainty as to what thresholds of indifference there are.[62] In our RL example we can think of uncertainty measurements in the average reward attained on each task, as well as uncertainty as to the task weights.[63]

External uncertainties refer to the lack of knowledge as to the consequences of a decision [194]. In our RL example, this may refer to the lack of knowledge as to the effects (which the decision-maker cannot control) of implementing a certain algorithm on a real-world task. For instance, the circumstance in which a hardware component fails may cause uncertainty in our decision. Including the measurement of the level of risk as a criterion for evaluation may resolve some of the above uncertainties.

A researcher may also want to identify all relevant **stakeholders** in their project as well as their roles and desires [26]. In the example of evaluating various deep RL algorithms, three separate stakeholders may include the algorithm designer, the algorithm evaluator and the end user. Whiteson et al. [12] propose the separation of these roles due to the assumption of a self-interested designer.

Finally, any **environmental constraints** should be identified [26]. This may include things like computational limitations as well as time and budget constraints which have significantly tangible impact on research. The environmental computational limitation constraint is particularly pertinent in this dissertation, as described by Jordan et al. [21] and Agarwal et al. [23]. This constraint limits our ability to have large sample sizes to evaluate over which would alleviate the need for strategies like bootstrapping and would allow for the use of cross-validation as in supervised learning. The alleviation of this constraint would also allow for the use of performance percentiles, as seen in Jordan et al. [21] in section 2.2. These would allow for the effective normalisation of performance scores while accounting for the non-linear scaling of performance scores with task difficulty. It is therefore, precisely, the limitation our computational capacity which has lead to us turning to MCDA to assist in resolving the problem of finding an effective normalisation technique.

Having reviewed the first stage of the MCDA process, problem identification and structuring, we turn to the next stage, model building and preference modelling. As we will see, the construction of value functions allows the evaluator to account for a decision-maker's preferences for different levels of performance. This will provide a means to overcome our problem of achieving a normalisation method that accounts for the non-linear scaling of performance scores with a decision-maker's preferences.

2.4.3 Model Building and Preference Modelling

Preference modelling consists of two primary components [26]: the preference models in terms of each individual criterion and an aggregation model. The preference models describe the relative desirability of achieving different levels of performance. This therefore deals with our aforementioned problem of how preference for certain performance levels on an RL task[64] scale in a non-linear manner with the performance scores. The aggregation model allows for combining of the preference model scores, across criteria, to allow for a multi-criteria comparison. MCDA will address our second problem of selecting weights so as not to bias a certain task when aggregating across tasks. This occurs whilst also taking into account the different scales of the tasks' preference models and taking into account the decision-maker's views on trading-off performances on one criterion for performance on another.

There are three broad schools of preference modelling according to Scott [25], Belton and Stewart [191] and Stewart [26]. These are (i) value measurement approaches, (iii) satisficing approaches and (iii) outranking approaches. Here, we provide a brief review of each.

[62] We discuss this below in terms of the weights used to aggregate performance scores across tasks and their interpretation as trade-off constants.

[63] which we use to aggregate across tasks.

[64] like Super-Mario [21]

2.4.3.1 Value Function Methods

Value function methods can be referred to as multi-attribute value theory (MAVT) approaches [25].[65] Here, the preferences of a decision-maker are reflected in the form of a value function, $y(k)$, which measures the value of alternatives $k \in \mathcal{L}$.[66] We will use the notation $g_j(k)$ to refer to the value of alternative k on criterion j.[67] Note that alternative k_1 is preferred to alternative k_2 for criterion j, $k_{1,j} \succ k_{2,j} \iff g_j(k_1) > g_j(k_2)$ [26]. Moreover, if $k_{1,j} \succcurlyeq k_{2,j} \forall k_{2,j} \in \mathcal{L}$, but $k_{1,j} \succ k_{2,j}$ for at least one $k_{2,j} \in \mathcal{L}$ then k_1 is said to *dominate* k_2 on task j. If no existing alternative k can dominate some alternative k^*, then k^* is said to be *Pareto optimal* [26]. Value function methods ascribe numerical, real numbered values to alternatives in order to ascertain some preference ordering of the alternatives. This implies that $y(k) : k \longrightarrow \mathbb{R}$ and that, $k_1 \succ k_2 \iff y(k_1) > y(k_2)$ and $k_1 \sim k_2 \iff y(k_1) = y(k_2)$.

Value functions allow for a complete weak ordering of preferences whereby [25, 26]: (i) Preferences are complete meaning that for two alternatives $k_1, k_2 \in \mathcal{L}$ either $k_1 \succ k_2, k_2 \succ k_1$ or $k_1 \sim k_2$ and (ii) preferences and indifferences are transitive whereby for three alternatives $k_1, k_2, k_3 \in \mathcal{L}$ if $k_1 \succ k_2$ and $k_2 \succ k_3 \implies k_1 \succ k_3$.

Partial Value Functions

In MCDA each criterion, j, is associated with some measurable attribute, $x_j(k)$, for which a partial preference function, $g_j(x)$ can be constructed [26]. These must hold the property that if $k_{1,j} \succ k_{2,j} \iff x_j(k_1) > x_j(k_2)$, meaning the properties of transitivity and completeness must hold. Note that if the preference functions in terms of individual criteria can be specified (trivially) in terms of scores attained on each criterion (and hence where each criterion is associated with a measurable performance level that is objectively quantifiable and implies an objective complete weak ordering of preferences) then this trivial selection of preference model can lead to misleading or biased results [26]. It is important to construct the preference model based on *value judgements* as preferences are rarely linearly related to the measurable performance levels of the criteria.[68] For instance in the case of RL, there may be problem-specific levels of reward above which further improvements yield little additional value. Similarly, there may be a performance threshold below which an algorithm is rendered completely useless. These non-linearities can have substantial effects on the solutions [26].

In general, preference models for each individual criterion, j, can be specified in terms of *partial value functions* [26]. We denote these as $g_j(x) = g_j(x_j(k))$. These must satisfy the transitivity and completeness properties of value functions in general in order to allow for an unambiguous ordering of alternatives. Importantly, MAVT says that these partial value functions must model the decision-maker's strength of preference.[69] The partial value functions must, therefore, also be non-decreasing but not necessarily linear functions. The properties required of the partial value functions are interrelated with the form of aggregation used.

Aggregation Across Criteria

The aggregation model component of value function methods has that, given partial value functions that satisfy the completeness and transitivity properties, if (and only if) all criteria are preferentially independent then there exists some additive value function [25, 193]. This can take the form, similar to Equation 20, found in Jordan et al. [21]. Moreover the additive value function may include weights $q_j \geq 0$ having $\sum_{j \in \Theta} q_j = 1$ such that [25]:

[65] This is also known as multi-attribute utility theory [26].
[66] Algorithms $k \in \mathcal{L}$ in our case.
[67] Tasks $j \in \Theta$ in our case.
[68] We discuss methods of value judgements in section 2.4.4 on constructing partial value functions.
[69] for different levels of performance (reward).

$$y(x_{j_1}(k), ..., x_{j_M}(k)) = \sum_{j \in \Theta} q_j \left[g_j \left(x_j(k) \right) \right] \tag{25}$$

Note that the addition or subtraction of a constant term to an additive value function will not affect the preference ordering yielded by $y(k)$ [25, 26]. Moreover, there is only an arbitrarily defined zero value for these functions. Hence, ratios of partial values $g_j(x_j(k))$ will not be meaningful in general.[70] Only ratios of differences between partial value functions $g_j(k)$ will have any absolute meaning. This defines the *interval scale* property of the value functions.[71]

Value function methods therefore place limited constraints on the preference structures being modelled for there to be an additive value function model [26]. Criteria need to be preferentially independent, partial value functions need to be of the interval scale and the conditions of transitivity and completeness need to hold. There are various ways to design value functions to elicit scores $g_j(x_j(k))$, for instance, the use of expected utility or preference modelling [25, 194]. We discuss these below, along with the matter of eliciting weights for our criteria. But before we do that, we briefly mention two other preference modelling techniques.

2.4.3.2 Satisficing Methods

Satisficing methods offer an alternative to value function methods. These do not attempt to explicitly model a decision-maker's preferences but instead attempt to satisfy certain levels of adequacy as specified by the decision-maker [25, 196]. These can be specified as goals as is the case for *goal programming* which is a variation of linear programming [25, 26]. Here, weights w_j^- and w_j^+ are optimised in order to minimise the negative and positive deviations from the various objectives $j \in \Theta$, respectively. The objective function minimised is then: $\sum_{j \in \Theta} \left(w_j^- d_j^- + w_j^+ d_j^+ \right)$. Satisficing methods are, therefore, useful to screen large sets of alternatives [25, 191].

2.4.3.3 Outranking Methods

Outranking methods focus on comparing alternatives in order to determine the amount of evidence to support the conclusion that one alternative is preferred to another [25]. A *concordance index* captures the strength of evidence that one alternative outranks another. For example, with the *ELECTRE I* method, modellers make use of the following measure of concordance for alternatives k_1 and k_2: $c(k_1, k_2) = \frac{1}{W} \sum_{j: x(k_1,j) \geq x(k_2,j)} w_j$, with $W = \sum_{j \in \Theta} w_j$ [25]. Here, w_j provide weights for each of the criteria, $j \in \Theta$. Veto levels can be set for each of the criteria such that if k_1 is outranked by k_2 on a certain criterion, by a certain amount, then it cannot outrank k_2 overall. Outranking methods are useful for identifying a subset of alternatives which are *incomparable* from one another, while excluding all other alternatives [25]. These methods are also useful since they are not affected by changes in units between criteria.[72] The methods are not necessarily useful in providing explicit performance measures, which concerns our interest in this dissertation.

The outranking and satisficing methods serve as reminders to the reader of alternatives methods for the selection of methods to evaluate alternatives. These methods are suitable based on the desires of the researcher and decision-maker. Considerations of alternative methods to evaluate the decision-maker's preference may prove useful in some RL contexts. Despite this, in this dissertation we focus on the value function methods only, as our interest is primarily concerned with the measurement of performance.

[70] except in specific circumstances where a zero-point exists [26].
[71] Hence we cannot claim that the performance of alternative k_1 is twice that of k_2, for instance.
[72] This may be particularly useful in a RL context.

2.4.4 Construction of Partial Value Functions

Turning to value function methods, recall that use of the additive value function model, described by Equation 25, requires that criteria, $j \in \Theta$ be complete, non-redundant and preferentially independent [195]. Additionally we require that partial value functions, $g_j(x(k))$, hold the properties of completeness, transitivity whilst being of the interval preference scale [195]. This is such that in the case where alternative k_1 is preferred to k_2 on task j_1, with k_2 preferred to k_1 on task j_2 and with, $q_{j_1}\left[g(x_{j_1}(k_1)) - g(x_{j_1}(k_2))\right] = q_{j_2}\left[g(x_{j_2}(k_2)) - g(x_{j_2}(k_1))\right]$ then, the gain made when selecting alternative k_2 on criterion j_2 is exactly compensated for by the loss on criterion j_1 when selecting k_2 instead of alternative k_1. Hence the use of this additive model implies that such a trade-off is independent of the absolute performance levels of the alternatives on the criteria [195].[73] Overall, the transformation of scores x_j, with partial value functions g_j, accounts for the non-linear scaling of a decision-maker's preferences with the measurable performance scores. This means that RL reward functions do not need to have changes in measurable performance correspond to equally challenging changes in difficulty for us to be able to aggregate across tasks.

Selecting partial value functions that abide by the above properties thus resolves the aforementioned problem of the non-linear scaling of task difficulty, which plagues the popular use of min-max normalisation functions such as those seen in Bellemare et al. [27] which look like $g(x_j(k)) = \frac{x - \min_k x_j(k)}{\max_k x_j(k) - \min_k x_{k,j}}$ for average scores $x_j(k)$ of algorithms k on task j. Moreover, Stewart [195] notes that a common misuse of additive value functions often occurs when a criterion is naturally available via some measurement x_j.[74] While it is permissible to use the measurements x_j in the MCDA process, the value needs to be transformed using some function $g_j(x)$, that abides by the interval scale property. Stewart [195] notes that, unfortunately, it is common for the above min-max transformation to be used which does not (in general) represent an interval scale of preferences. This is since changing rates of trade-offs between criteria, as x_j varies between its minimum and maximum, is the norm rather than the exception, especially if the range between minimum and maximum is large. Furthermore, Stewart [195] notes that such inappropriate linear transformations can lead to favouring of extreme outcomes. For instance it can lead to the favouring of very good performances on some tasks and very poor on others, over more balanced solutions. Stewart [195] cites simulation studies [197, 198] that demonstrate that the largest source of bias in reported results that use additive value function models, can result from such inappropriate transformations. These studies [197, 198], however, demonstrate that the use of *piecewise linear approximations* to partial value functions $g_j(x_j)$ largely eliminates the bias. These are the methods we turn to now for constructing our partial value functions.

Partial value functions reflect the scoring process of MCDA. This refers to the assessment of the value derived by the decision-maker from the performance of an alternative on a certain criterion. Since the partial value functions must be scored on an interval scale, the *difference* between points is the important factor [195]. Hence to construct a value scale, two reference points need to be established, with numerical values allocated to each. It is typical to use the best and worst possible points for a criterion as the reference points, with values of 0 and 100 assigned to these, for instance. This gives $g_j(x_{min}(k)) = 0$ and $g_j(x_{max}(k)) = 100, \forall j \in \Theta, k \in \mathcal{L}$.

These reference points can be defined *locally* using the best and worst possible performance scores in the set of alternatives such that $x_{min} = \min_k x(k)$ and $x_{max} = \max_k x(k)$ [195]. They can also be defined *globally*, using the best and worst possible scores for a task. The choice here depends on the availability of either local or global information. Global scales have the advantage of being able to be defined before the acquisition of the performance scores of the alternatives have been obtained [195].

Partial value functions can be assessed directly, often using visual methods, or indirectly [195]. von Winterfeldt and Edwards [192] note that if the problem has been well structured then the function should have no discontinuities and be monotonic. For a direct assessment of partial value function, the next step in constructing the function is to decide whether a monotonic increasing or monotonic decreasing function

[73] For instance, the same compensation would occur for $g_{j_1}(x_1) = 5$ and $g_{j_1}(x_2) = 0$ as with $g_{j_1}(x_1) = 15$ and $g_{j_1}(x_2) = 10$.
[74] such as is the case for performance scores on RL tasks.

appropriately reflects the decision-maker's preferences for different performance levels [195]. Given the properties of a continuous and (relatively) smooth value function, value functions can then be sketched directly by an evaluator in order to reflect the preferences for the various levels of performance. This would usually result in a concave, convex or sigmoidal shape [195]. In less familiar settings a user may desire an indirect method.

Indirect methods assume a monotonic increasing or decreasing value function over the specified range defined by local or global reference points [195]. Indirect methods construct value functions by asking questions of the evaluator. Importantly, indirect methods look to elicit from the user points of *indifference* in terms of preference for different levels of performance [194, 197]. These methods of elicitation aid in a robust approach to constructing such value functions.[75] [76] The *bisection* method is one example of an indirect construction method for partial value functions.

2.4.4.1 Bisection Method for Partial Value Functions

The bisection method, as identified by von Winterfeldt and Edwards [192] offers an indirect method for assessing partial value functions. As per the example used by Stewart [195], we can think of the problem of selecting a city in which to establish a business office. We construct a partial value function for an identified criterion that measures the number of available staff for each post in the office. We obtain measurable information via recruitment agencies as to the number of qualified applicants per post available. The end points can be defined on a local scale, where we can expect at minimum 4 applicants per post and at most 50. This gives $x_{min} = 4$ and $x_{max} = 50$ We then identify that the partial value function will be monotonically increasing as preferential value increases with the number of applicants (level of performance).[77]

This indirect method then asks of an evaluator to identify the point on the performance scale that is the midpoint in terms of preference between the best and worst possible performance levels [195]. In other words the evaluator must find the point $x = x_{50}$ such that $g_j(x_{50}) = 50$. Continuing with our example, and after some thought, say the midpoint identified is 10 applicants. This means that the user would be indifferent between the change from 4 to 10 applicants as with a change from 10 to 50 applicants in terms of their preferences. This means that finding $x = x_{50}$ solves the problem of finding the point that gives $g_j(x_{max}) - g_j(x_{50}) = g_j(x_{50}) - g_j(x_{min})$.

The next step is for the evaluator to identify the midpoints in terms of preference between 4 and 10 applicants and between 10 and 50 applicants. This means finding x_{25} and x_{75} such that $g_j(x_{25}) = 25 \iff g_j(x_{50}) - g_j(x_{25}) = g_j(x_{25}) - g_j(x_{min})$ and $g_j(x_{75}) = 75 \iff g_j(x_{max}) - g_j(x_{75}) = g_j(x_{75}) - g_j(x_{50})$. According to simulation studies [197, 198] that look at the robustness of using the bisection method, it is adequate and generally acceptable to use as few as 5 reference points in the construction of a partial value function [195]. This means that the 2 endpoints and 3 midpoints should give robust enough information for an evaluator to sketch a piece-wise linear value function. Given the 5 points to be $\{x_{min}, x_{25}, x_{50}, x_{75}, x_{max}\} = \{4, 8, 10, 20, 50\}$ then Figure 11 depicts such a partial value function [195].

2.4.4.2 Difference Methods for Partial Value Functions

Two more indirect methods of constructing partial value functions are established by Watson and Buede

[75] While constructing partial value functions is a subjective procedure, if one conducts these specifically designed elicitation methods, does so iteratively, with a domain expert, and performs consistency checks with the expert, then this will aid in constructing a robust partial value function [194].

[76] In this dissertation we suggest the necessity to construct partial value functions by eliciting information from users who have expert-level knowledge about the RL tasks being used. This can be done with multiple experts where a consensus for which of their constructed value functions to use is work-shopped before being selected [194].

[77] Note that this happens in a non-linear way.

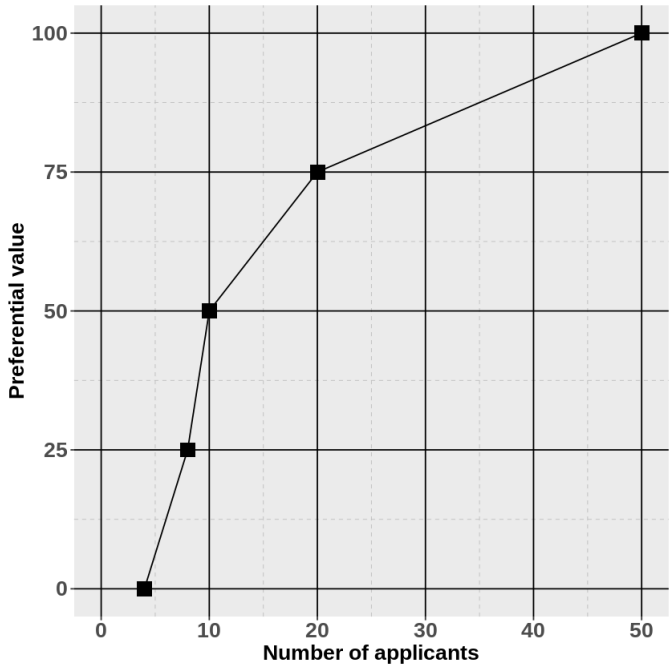

Figure 11: This figure depicts the construction of a partial value function measuring the strength of preference of a user for the different numbers of available staff for an advertised role. The figure can be found in an example in Stewart [195]. The decision-maker has constructed this value function using an indirect value function method called the bisection method [192]. They have identified the endpoints on a local scale, being that the minimum possible number of available staff is 4 and the corresponding maximum is 50. They have then used the bisection method to assess the 3 midpoints, in terms of preference between the endpoints. This has allowed a piecewise linear construction of the partial value function. The function expresses value on the scale $[0, 100]$.

[199] and by von Winterfeldt and Edwards [192]. These can be referred to as *difference* methods [195].

The difference method used by Watson and Buede [199], takes the measurement range of the criterion in question and divides it into intervals of equal length. The evaluator then rank orders the intervals in terms of which swing, from lowest to highest performance level, in an interval is most preferred (highly valued) [195]. Following an example by Stewart [195], we investigate the case where a MCDA process is assessing a criterion which measures accessibility to a company's headquarter office. This is measured in terms of the number of flights from various cities across the world per week. The alternatives in question are therefore each of the cities which have flights that connect to the headquarter city. The criterion measures the number of flights per week for each alternative. The evaluator establishes a global reference scale with the minimum possible number of flights per week being $x_{min} = 0$ and the corresponding maximum as $x_{max} = 28$. Furthermore, we note that there is a preference for an increase in the number of flights meaning that we will have a monotonic increasing and relatively smooth value function to construct. The evaluator then divides the criterion's measurement range into 4 equal intervals of length 7 and rank orders the intervals in terms of their preference for swings in the performance levels from best to worst for each interval.

In other words, starting with an interval $A = [0, 28]$, we divide A up to form the collection of 4 intervals $\mathcal{A} = \{A_\alpha, \alpha \in I\}$ with $I = \{1, 2, 3, 4\}$. This is such that $\bigcup_{\alpha \in I} A_\alpha = A$, $\mathcal{A}$ are pairwise disjoint[78] and $|A_\alpha| = 7$ for $\alpha \in I$. This gives $\mathcal{A} = \{[0, 7], [7, 14], [14, 21], [21, 28]\}$. Table 3 demonstrates the example whereby the decision-maker's preferences result in an ordered rank ordering of the intervals in $\mathcal{A}$. The ranking demonstrates the strongest preference for the swing from worst to best of the interval $[0, 7]$ and the weakest preference for the swing across interval $[21, 28]$. Hence, if we denote the interval endpoints with

[78] except for overlapping endpoints.

$A_\alpha = [a_{\alpha,\min}, a_{\alpha,\max}]$ for $\alpha \in I$ then, the decision-maker's preferences have produced a ranking such that $g_j(a_{\alpha_1,\max}) - g_j(a_{\alpha_1,\min}) > g_j(a_{\alpha_2,\max}) - g_j(a_{\alpha_2,\min})$ for $\alpha_1 < \alpha_2, \forall \alpha_1, \alpha_2 \in I$.

Table 3: A decision-maker's ranked preferences for the intervals reflecting an increase in number of flights from some city to the decision-maker's headquarter office city. The rankings reflect which interval increase they prefer the most and value the highest. This example can be found in Stewart [195] and reflects a difference method by Watson and Buede [199] for constructing partial value functions. An example of a possible resulting partial value function is sketched in Figure 12.

Increase in Number of Flights		Increase in Value
From	to	
0	7	1 = Greatest increase in value
7	14	2
14	21	3
21	28	4

The established ranking allows the evaluator to ascertain an idea of the concavity of the partial value function, i.e. that we will have a concave, increasing function, similar to that seen in Figure 12. The concavity is determined by the gradients of the function on each interval. The highest ranked interval will have the largest gradient and, correspondingly, the lowest ranked interval will have the smallest gradient, reflecting the decision-maker's preferences. Stewart [195] suggest that the function can be sketched *directly* based on this information alone, or refined by asking the decision-maker to assess the relative magnitude of each value increase for each interval.

Instead of dividing up the interval A into a collection of intervals, $\mathcal{A}$, and evaluating the different increases across the intervals A_α, von Winterfeldt and Edwards [192] suggests a second difference method which offers a reverse procedure. The method by von Winterfeldt and Edwards [192] suggests choosing an interval endpoint $a_{1,\max}$ on the criterion's performance scale, that defines the interval $A_1 = [x_{min}, a_{1,\max}]$ such that the increase across the interval, $g_j(a_{1,\max}) - g_j(x_{min}) = u$, defines an arbitrary unit level of value u. Then one must find the endpoints $a_{\alpha,\min}$ and $a_{\alpha,\max}$, $\alpha \in I, I = \{1, 2, ..., \zeta\}$ which define intervals $A_\alpha = [a_{\alpha,\min}, a_{\alpha,\max}]$ such that $\bigcup_{\alpha \in I} A_\alpha = A$, the A_α are pairwise disjoint[79] and $g_j(a_{\alpha,\max}) - g_j(a_{\alpha,\min}) = u$. This then has that $a_{\alpha,\max} = a_{\alpha+1,\min}$. For this procedure, the endpoint $a_{1,\max}$ is chosen to be between one fifth and one tenth of the difference between the maximum and minimum. This gives $a_{1,\max} = \kappa(x_{max} - x_{min})$ for $\kappa \in [\frac{1}{10}, \frac{1}{5}]$.

Continuing with the number of flights example from above [195], one could define a unit level of value by choosing $a_{1,\max} = 3$ flights per week which is almost a tenth of the range ($\kappa \approx \frac{1}{10}$). Given the interval $A = [x_{min}, x_{max}] = [0, 28]$, this gives $A_1 = [0, 3]$ and $u = g_j(3) - g_j(0)$. The decision-maker is then asked to find $A_2 = [a_{1,\max}, a_{2,\max}] = [3, a_{2,\max}]$ by naming the number of flights $a_{2,\max}$ such that an increase in value by moving across A_2 is equivalent to u and, therefore, $g_j(a_{2,\max}) - g_j(3) = u = g_j(3) - g_j(0)$. This means that the decision-maker is asked to name the number of flights $a_{2,\max}$ for which an increase in value from 0 to 3 flights is equivalent to an increase in value from 3 to $a_{2,\max}$ flights.

The same question is then repeated for some number of flights $a_{3,\max}$ to find $A_3 = [a_{2,\max}, a_{3,\max}]$ having $g_j(a_{3,\max}) - g_j(a_{2,\max}) = u$. This process is repeated until $a_{n,\max} = x_{max} = 28$. These responses are reflected by the value function in Table 4. This function is then plotted in Figure 12.

[79] except at endpoints.

Table 4: Partial value function specifying the user's preference for the number of flights from any city to the user's headquarter office city. This example can be found in Stewart [195] and reflects a difference method by von Winterfeldt and Edwards [192] for constructing partial value functions. The function is sketched in Figure 12. The unit value is found to be $u = 25$.

Number of flights	Value (per units defined above)	Value (0 to 100 scale)
0	0	0
3	1	25
7	2	50
14	3	75
28	4	100

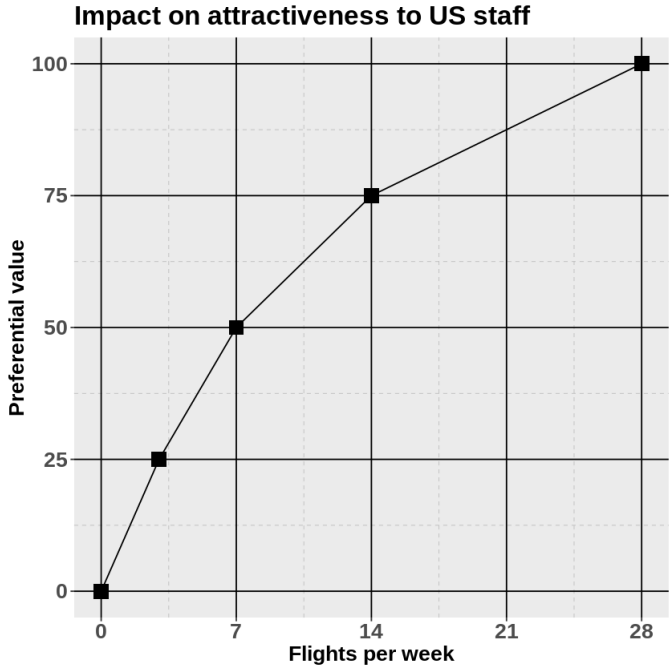

Figure 12: This figure depicts the construction of a partial value function measuring the strength of preference of a user for the different numbers of flights available from any city to a company's headquarter city. The figure can be found in an example in Stewart [195]. The decision-maker has constructed this value function using an indirect value function method called a difference method [192]. They have identified the endpoints on a global scale, with the minimum number of flights available being 0 and the maximum being 28 per week. They have then used the difference method by von Winterfeldt and Edwards [192] which specifies a unit level of value to be the user's value of an increase from 0 to 3 flights. The method then asks the decision-maker to specify the intervals where the difference between an interval's end-points (in numbers of flights) is equivalent in preferential value to the value of the difference between 3 and 0 flights. This function is defined by Table 4. The function expresses value on the scale $[0, 100]$.

2.4.5 Weights and Aggregation for Value function Methods

With several methods for constructing partial value functions we now have a means of normalising our performance scores for each RL algorithm on the different tasks. Furthermore we have a means of normalising the scores which takes into account the non-linear scaling of difficulty with performance scores[80] Using the partial value functions we now need to account for the second problem identified with common RL algorithm performance evaluations, a biased weighting of tasks in the aggregation of performance scores across tasks.

[80] and which does not suffer from clustering in regions of [0,1].

As we seek to aggregate across tasks, we seek a weighting of the tasks which does not bias a particular task unless it is deemed more important for the performance evaluation. We hence consider the assessment of weights q_j, seen in Equation 20 and, again, in Equation 25. It is important to note that one can consider a value tree with more than one level of criteria for decision-making. For instance, consider the case in which there are two hierarchical levels of criteria in the value tree, then the aggregation component of preference modelling may occur in a hierarchical manner [26]. This is whereby, first aggregation occurs at the lower level of the value tree, across criteria that share the same parent criterion. Then aggregation over the higher level criteria will occur. Say we have M upper level criteria (for instance our $M = |\Theta|$ tasks in our RL scenario) and N lower level criteria per upper level criteria (for instance our N runs per task in our RL scenario), then the overall value function can be expressed as [26]:

$$y(k) = \sum_{j\in\Theta}\sum_{n=1}^{N} q_{j,n}\left[g_{j,n}(x_{j,n}(k))\right] = \sum_{j\in\Theta} q_j \sum_{n=1}^{N} q_{n|j}\left[g_{j,n}(x_{j,n}(k))\right] \tag{26}$$

In this case q_j is the measure of the importance of the j^{th} upper level criterion. Moreover, $q_{n|j} = \frac{q_{j,n}}{q_j}$ is the importance of the lower level criterion n, relative to the other sub-criteria of the same upper level criterion j [26]. In the case of our RL algorithm evaluation example, we weight each of the lower level criteria equally as they correspond to runs on the same task, which will have the same partial value functions. This gives $q_{n|j} = \frac{1}{N} \implies q_{j,n} = \frac{q_j}{N}$ and we are left with equivalence to Equation 20:

$$y(k) = \sum_{j\in\Theta}\sum_{n=1}^{N} q_{j,n}\left[g_j(x_{j,n}(k))\right] = \sum_{j\in\Theta} \frac{q_j}{N} \sum_{n=1}^{N} \left[g_j(x_{j,n}(k))\right] \tag{27}$$

To bring further meaning to the weights, q_j, we note that the weights in such an additive value function can act as scaling constants [26]. This is in such cases where the different value scales of the criteria need to be scaled compatibly for aggregation. For instance, given two partial value functions for two criteria that satisfy the properties of preferential independence and are of the interval scale, then there exists an additive value function, $y(k)$. Then suppose that for two alternatives, $k_1 \sim k_2 \implies y(k_1) = y(k_2)$. Suppose then that these two alternatives still differ for two criteria j_1 and j_2 such that $g(x_{j_1}(k_1)) \neq g(x_{j_1}(k_2))$ and $g(x_{j_2}(k_1)) \neq g(x_{j_2}(k_2))$. Then [26]:

$$\begin{aligned}
&\implies q_{j_1} g(x_{j_1}(k_1)) + q_{j_2} g(x_{j_2}(k_1)) = q_{j_1} g(x_{j_1}(k_2)) + q_{j_2} g(x_{j_2}(k_2)) \\
&\implies q_{j_1}\left(g(x_{j_1}(k_1)) - g(x_{j_1}(k_2))\right) = q_{j_2}\left(g(x_{j_2}(k_2)) - g(x_{j_2}(k_1))\right) \\
&\implies \frac{q_{j_1}}{q_{j_2}} = \frac{\left(g(x_{j_2}(k_2)) - g(x_{j_2}(k_1))\right)}{\left(g(x_{j_1}(k_1)) - g(x_{j_1}(k_2))\right)} \\
&\implies \frac{q_{j_1}}{q_{j_2}} = \frac{\Delta g_{j_2}}{\Delta g_{j_1}}
\end{aligned} \tag{28}$$

for $\Delta g_j = \left(g(x_j(k_2)) - g(x_j(k_1))\right)$. The assessment of weights must then be consistent with this requirement [26]. The weights thus represent trade-off information between criteria and act as scaling constants [25]. Thus if criterion j_1 has a weight which is half that of criterion j_2 then this can be interpreted as a decision-maker valuing 10 value points on criterion j_1 as trade-able with 5 value points on criterion j_2 [195].

The weights assigned to partial value functions are determined by the scales used to measure these functions [25]. Furthermore, partial value functions can be re-scaled as long as their corresponding relative weight is re-scaled. Lastly, if partial value functions are re-scaled by a constant factor then this does not affect the ordering of alternatives. Hence, only ratios of weights have absolute meaning.

In order to assess the weights, and since they are determined by the scale of the partial value functions,

it is important to select reference points such as the best and worst points on performance scales [25].[81] Weights can then be assessed using the *swing weighting process* proposed by von Winterfeldt and Edwards [192].

The swing weighting process [192] assesses how much of an increase in a criterion is *trade-able* (according to the decision-maker) for an increase of one unit in a second criterion. The increase in one unit can refer to a *swing* from the worst to the best performance score. To assess this one asks of the decision-maker to first consider a scenario where an alternative performs in the worst case for all criteria in Θ [25]. The decision-maker then must select their preference from $M = |\Theta|$ options, where in each option the alternative performs in the worst case for all but one of the M criteria, and in this one criterion, say criterion $j \in \Theta$, the alternative performs at its best. The option chosen by the decision-maker gives the weight q_{j_1} as the largest weight.

Subsequently, the decision-maker must consider $M-1$ options where criterion $j \in \{\Theta - j_1\}$ performs at its best, say $g_j(x_j(k)) = g_j(x_{max}) = 100$, denoting the optimal performance [25]. In each of these $M-1$ options the criteria performances will all be set to their worst case, except for one which will be set to its optimal performance. The decision-maker then selects their preference for which of the $M-1$ criteria they prefer to be at the optimal level. This yields the second largest weight. This process is iteratively repeated until a rank ordering of preferences for swings in the criteria are established by the decision-maker.

The process of assigning values to the weights can be decided by requesting of the decision-maker to estimate the relative importance of the pairs of swings $\frac{q_{j_1}}{q_{j_2}}$ [25]. For instance if, q_{j_2} is the largest weight then we have ratios, $\frac{q_{j_i}}{q_{j_2}} < 1$ for $j_1 \neq j_2$. If we set, $q_{j_2} = 100$ then the values of weights $q_{j_i} \forall j_i \in \{\Theta - j_2\}$ can be determined by asking the decision-maker [195]: *What is the relative value of a swing from worst to best for the other criteria as compared to a value of 100 for the highest ranked criterion?*[82] This process requires $M-1$ comparisons per family of M criteria[83]. This approach offers the most tractable assignment of weights [25]. However, in order to ensure consistency, a number of methods require a higher number of checks such as the $\frac{M(M-1)}{2}$ checks required by the *AHP* and *MACBETH* methods [25, 200]. The minimal number of checks is proposed by Stewart [195], by specifying a reference criterion and having all other criteria cross-checked against this.[84]

The theoretically most defensible way to select weights, according to Scott [25], is to request that the decision-maker specify indifference points, where for two alternatives, $y(k_1) = y(k_2)$ but for criteria j_1 and j_2, $g(x_{j_1}(k_1)) \neq g(x_{j_1}(k_2))$ and $g(x_{j_2}(k_1)) \neq g(x_{j_2}(k_2))$. Then weights can be determined using Equation 28.

It is noteworthy to mention that if an intrinsically important criterion has its minimum and maximum points on the value scale correspond to similar levels of performance then its weight may be ranked quite low [195]. It is also noteworthy that the use of alternative reference points such as *neutral* and *good* reference points can replace the use of the best and worst reference points [195].

Given a set of standardised weights, where the largest weight is 100, it is common to normalise these to sum to 1 [195]. This helps decision-makers to interpret the normalised weights in terms of their contributions to the total value. The original (standardised) weights reveal only the relative importance as compared to the largest weight.

The above mentioned weights are determined within families in the value tree [195]. This means that the

[81] As mentioned earlier, these can be assessed in a local manner by selecting the best and worst amongst the set of alternatives or a global manner, whereby the best and worst possible performances are assessed [25]. There is no fundamental reason for selecting either a local or global scale [25].

[82] This process can be done visually, whilst removing the need for numerical precision [195].

[83] In ranking the criteria to assess the weights for top-level criteria (within families) the evaluator must note that the swing weight method considers swings in all sub-level criteria (from worst to best) [195].

[84] Consistency checks like this, using the swing weight method and performing sensitivity analyses, aid in the robustness of the selection of swing weights.

swing weight comparisons are typically done for siblings under the same parent within a value tree. These weights (within a family) are then normalised to sum to 1. The weights within a family are called *relative weights*. One can also define cumulative weights which are equal to the product of a relative weight (in comparison with its siblings) with the relative weight of its parent, and the relative weight of its parent's parent, and so on to the top of the tree. Cumulative weights appear in Equations 21 and 26 as $q^*_{n,j}$. These have the property that, by definition, the cumulative weights of all leaf criteria sum to one. Hence, for a small enough tree with few leaf-level criteria, it may be most efficient to simply elicit weights using the swing weight method for leaf-level criteria only[85].

Using the above, we have a method for selecting weights $q_j \; \forall j \in \Theta$ and can use these along with our partial value functions to construct a value function for our MCDA problem as per Equation 26. Furthermore, our conceptualisation of *swing weights* captures both the concept of the relative *importance* of each criterion with respect to one another and the extent to which the measurement scales used discriminate between criteria [195].

2.5 Section Summary

In the above section, we began by defining the context of this dissertation: the RL problem with some of its solution methods. We then extended this to the multi-agent setting to define the cooperative MARL problem, with some of its solution methods. We separated our work on performance evaluations from more theoretical analyses, focusing instead on empirical analyses. We then reviewed what the RL literature says about these.

This dissertation focuses on the dominant research and implementation paradigm at the moment: *deep* learning.[86] While we include notes on other aspects of empirical evaluation, such as reliability, we primarily focus on explicit measures of absolute performance as done in the RL literature. These works shine light on various problems affecting the lack of rigour and methodological standards in RL research. They also offer many suggestions and solutions which we reviewed.

We decided upon three main areas of review for our study of the RL evaluation literature: RL task selection, uncertainty measurement and the explicit measure of empirical algorithm performance. The latter involves stages of data collection and data aggregation which includes choices of an aggregation function and normalisation methodologies, which we discussed. Lastly, we reviewed some suggestions from the literature on result reporting and also some commentary on the extent to which algorithm implementation variance causes reported results between papers to vary.

We identified two problems with evaluation that still require addressing. These are that the normalisation functions used do not account for the non-linear scaling of algorithm performance scores with task difficulty and that the aggregation across tasks used provides a biased weighting of tasks. The identification of these problems responded to our first research question, which questioned what problems remain with RL evaluation and why these are an issue.

We then drew similarity between the evaluation of deep SARL and cooperative MARL empirical algorithm performances, noting their equivalence. We highlighted some MARL literature which discusses how MARL performance evaluations suffer from the same problems as SARL evaluations. We mentioned a few examples in the MARL literature of new benchmark performance assessments and implementation optimisations, but noted the lack of prevalence of reviews of the state of MARL evaluation procedures. Hence, we instead analysed a dataset from Gorsane et al. [1].

We decided to analyse this dataset because of the problem equivalence between SARL and cooperative

[85] In this case the relative weights can be calculated by normalising the cumulative weights (to sum to 1) within families of nodes within the value tree.

[86] As traditional tabular methods have fallen out of favour.

MARL evaluations. Cooperative MARL can thus be considered a subset of RL research, and specifically a *recent* subset. Therefore, we analysed this recent subset of RL evaluation in terms of its level of rigour, given the lack of similar analyses in this area. This case study, therefore, provided us with insights into the state of RL evaluations in the literature in general. Additionally, due to the problem equivalence, we can take the lessons learnt here to the broader RL field.

The dataset was collected from many recent deep MARL papers since 2016 and consisted of evaluation methodologies used in the cooperative MARL literature. This dataset was examined via an exploratory data analysis which revealed large spreads in reported algorithm performances across papers (Figure 7), similarly to SARL. The study of the dataset also revealed that very few tasks were being used for evaluations (Figure 9) and that there were major inconsistencies in the evaluation methodologies used across the literature (Figure 10). This exploratory data analysis served to provide evidence toward confirming our hypothesis of a poor level of rigour in the empirical performance evaluations of deep cooperative MARL algorithms. This hypothesis was established at the beginning of this dissertation to address our second research question on diagnosing the level of rigour in deep cooperative MARL research. The hypothesis stemmed from the literature on deep SARL performance evaluations and the works which noted similar inconsistencies in MARL research, as well as the equivalent characteristics between evaluations in SARL and cooperative MARL.

The evidence from the exploratory data analysis lead us to continue addressing the performance evaluation procedures in RL. Hence, we reviewed the MCDA framework for decision-making. Here, we noted that we can view the decision of choosing an algorithm from a set of algorithms as a multi-criteria decision problem. The review of MCDA looked at a system for problem structuring and preference modelling.

Problem structuring established the algorithms as a set of alternatives to choose between, and the set of RL tasks as a set of criteria upon which to evaluate the alternatives. Thereafter, preference modelling afforded us the ability to capture a decision-maker's preferences for different performance levels on each criterion within partial value functions. These need satisfy a set of properties allowing for an additive value function to be used for the aggregation across multiple criteria.

Importantly we established that criteria need be measurable and satisfy the property of preferential independence. Furthermore, value functions need satisfy the properties of transitivity and completeness and be of the interval scale. This guarantees the existence of an additive value function. Moreover, weights need satisfy the trade off property in order to capture both the relative importance of a criterion with respect to other criteria and the extent to which the measurement scales used discriminate between criteria. The weights used thus address the problem of a naive, uniformly weighted aggregation procedure leading to a bias toward one or more criteria as compared to the rest.

Furthermore, the problem of using some flawed normalisation function, which does not account for the non-linear scaling of a decision-maker's preferences with levels of performance is also addressed via the construction of partial value functions to capture the decision-maker's preferences. We reviewed the bisection method and two difference methods for capturing the decision-maker's preferences and constructing a partial value functions for each criterion. Pairing this process with the swing weight method for eliciting the weights of the criteria, we can now construct our additive MCDA value function.

Using MCDA thus poses a potential path forward to address deep RL empirical algorithm performance evaluations. Hence we pose our third research question: *Can MCDA be used to benefit a guideline for coherent deep RL empirical algorithm performance evaluations, and if so, how?* Here, we hypothesise the effective use of MCDA based on its simple and coherent guidelines and its ability to resolve the two evaluation problems remaining in the literature: (i) flawed normalisation procedures and (ii) flawed weighting of task scores when aggregating across tasks. In summary, having defined and diagnosed the problem of empirical evaluation in deep RL, we proceed in the next section to take the lessons from the RL evaluation literature and MCDA framework to establish a guideline for better empirical algorithm performance evaluations.

3 Constructing a Guideline for Evaluation

In this section we aim to resolve one of the primary research questions in this dissertation. We look to resolve the question of whether MCDA can be used to provide a coherent empirical algorithm performance evaluation guideline for deep RL. This guideline must consistently measure performance across various RL tasks and must also be consistent with the value function properties outlined in section 2.4. This section therefore outlines the construction of a guideline starting with subsection 3.1. Our guideline is then presented in subsection 3.2 and discussed in subsection 3.3.

3.1 Method: Producing a Guideline for Deep RL Empirical Algorithm Performance Evaluations

This subsection provides the methodology for addressing the central focus of this dissertation, addressing the performance evaluation procedures of deep RL. This subsection provides our method for compiling a guideline for such performance evaluations. We will primarily draw on the insights provided in the literature on evaluation in RL (subsection 2.2) as well as the insights of the MCDA framework (subsection 2.4) which provides a coherent way of choosing some alternative from a set of alternatives when multiple criteria are used for evaluation. This subsection provides the methodology to answer our third research question: *Can MCDA be used to benefit a guideline for coherent deep RL empirical algorithm performance evaluations, and if so, how?*

In response to this research question we hypothesise that MCDA is effective at enhancing a guideline for effective evaluation of deep RL algorithms. This hypothesis is based on MCDA being able to encompass the decision problem of selecting an algorithm from a set of RL algorithms using the performance evaluations on various RL tasks as criteria. This was demonstrated as an example across subsection 2.4. The hypothesis is also based on MCDA offering something that is lacking from RL evaluation methodologies: The elaboration of a performance measurement procedure which measures performance consistently across tasks whilst weighting the tasks in such a way as to not bias a certain subset of tasks in providing an aggregated performance measure.

These aspects of MCDA resolve the problems identified by our first research question which examines the problems that remain with RL evaluations. Through a literature review (in subsection 2.2) we discussed the problem of normalising the performance scores on individual tasks to provide an aggregated performance score across all tasks, and we discussed the problem of providing a non-biased weighting of individual task scores when performing such an aggregation. We elaborated on the use of MCDA resolving these issues in subsections 2.2 and 2.4.

As a reminder, our undertaking of this research method is also in response to the problem diagnosis of our second research questions which examined (through a cooperative MARL case study[87]) the state of the level of rigour of RL evaluations amongst recent publications. Our diagnosis was that of a poor level of consistency across research papers. This was observed through highly varied results of reported algorithm performance, varied numbers of tasks used to report performance per paper, and differences in the choices of aggregation functions and uncertainty metrics. This problem diagnosis therefore motivates our endeavour.

3.1.1 Problem Structuring

We begin the construction of our guideline using the first step in the MCDA procedure, problem structuring. As seen in subsection 2.4 this can be done using the *CAUSE* mnemonic standing for: criteria, alternatives,

[87]This research question was answered in subsection 2.3, via an exploratory data analysis using the dataset of evaluation procedures from Gorsane et al. [1].

uncertainties, stakeholders and environmental constraints. Subsection 2.4 briefly reviewed how these match to the specific case of algorithms in RL. Here we elaborate more on this problem structuring approach.

As MCDA stands for multi-criteria decision analysis we first establish what our decision problem is within the context of evaluating algorithm performances in deep RL. We could view our problem as a *choice problem* or a *ranking problem*, where we either seek to *choose* one alternative from a set of possible alternatives, or we seek to *rank* our set of alternatives. Here, we view our decision problem as looking at the *alternatives* as being the set of deep RL algorithms to choose from (or to rank). We denote the algorithms as k and the set of algorithms $\mathcal{L}$ such that $k \in \mathcal{L}$.

The *criteria* in our particular decision problem are the various deep RL tasks that the algorithm evaluator decides are appropriate for assessing the algorithms' capabilities. Each task is denoted by j in the set of tasks Θ selected by the algorithm evaluator. This is such that $j \in \Theta$. Recall from subsection 2.2 that algorithms should be tested on several tasks to avoid overfitting on any particular task or subset of tasks. These tasks should be selected so as to align with the algorithms' intended use. The tasks should reflect a complete and non-redundant set of criteria upon which to evaluate the algorithms [195]. Furthermore, the tasks should be measurable and preferentially independent [25] so as to comply with properties required later for additive aggregation.

Continuing with the problem structuring approach according to the CAUSE mnemonic, in our problem's context, we emphasise the importance of the measurement of *uncertainty* as an expression of the variability in reported aggregation metrics of the algorithms' performances. We also have uncertainty as to which weights we use in our additive aggregation across tasks [25]. Other uncertainties to consider in the modelling process include the uncertainty as to which tasks to include [195]. We note that the uncertainty measurements of our performance scores may be difficult to obtain due to a key *environmental constraint*: Our limited computational capacity.

The computational complexity of deep RL places a practical limitation on the amount of training and evaluation runs that are feasible to do when designing and testing algorithms. Because of this limitation it is necessary to measure performance within a small data regime, making use of the limited run data available. This issue was addressed by Agarwal et al. [23]. Hence it is appropriate to use their recommendations here. They address the measurement of uncertainty with respect to the environmental constraint of limited computational capacity by suggesting the use of *stratified bootstrap confidence intervals*. These are able to produce uncertainty estimates using only a *handful of runs*. We discuss this recommendation in section 2.2.

Importantly the measurement of uncertainty pertains to the individuals we refer to as the *evaluators*. This relates to the final idea in the CAUSE mnemonic, the *stakeholders*. In our decision problem's context these can be identified as the algorithm designers, the algorithm evaluators and the end users. Here, we are primarily focused on the role of the evaluators. The evaluators will be responsible for the selection of the RL tasks in which the algorithms will be tested. These will take into account the contexts in which the end user will use the respective algorithms. The evaluator will also be responsible for data collection and the explicit measurement of performance. These topics will be discussed next.

3.1.2 Data Collection

3.1.2.1 Algorithm Selection

The first step in the data collection process is to decide upon a set of algorithms $\mathcal{L}$ which will be used to gather data for your performance assessment or experiment. Perhaps one algorithm is of interest and is to be compared to a set of baselines. These baseline algorithms could be selected because they are (at present) commonly used in the literature, as is done with clinical drug trials. Each algorithm we choose is indexed by k such that $k \in \mathcal{L}$.

3.1.2.2 RL Task Selection For Evaluation

The next stage of data collection will be to select the RL tasks to be used for evaluating the performances of the algorithms. As discussed in section 2.2, the views of Whiteson et al. [12], Jordan et al. [21] and Agarwal et al. [23] recommend the selection of multiple tasks for evaluation. This assists with the prevention of overfitting [16, 74] as well as being a better test of algorithm generalisation and preventing the capacity for cherry-picking [1]. While no consensus on the methodology for specific task selection was decided upon we revert to the recommendation of selecting several tasks for evaluation. All of these tasks must be relevant to the potential use-cases of the algorithm, by the end-user. We also recommend a thorough process of review for the potential options of which tasks are relevant. Further variation in the initial states, random seeds, transition models and reward models may provide suitable variation in the tasks selected for evaluation. Careful consideration of which properties of the algorithms that these tasks may test should also be considered. A further consideration for the presently popular tasks and challenges should also be made. Environment suites such as SMAC [81] or MPE [58] could be considered [1]. Crucially, the tasks selected should attempt to form a complete set of criteria for the evaluation in mind [195]. The criteria should not, however, contain redundancies and be preferentially independent [195]. Finally, our denotation for task selection has that the selection of a number, M, of tasks j forms a set of tasks for evaluation, Θ, such that $j \in \Theta$ and $|\Theta| = M$.

3.1.2.3 Hyperparameter Tuning

Once the tasks for evaluating the algorithms are selected and the algorithms coded, then hyperparameters can be tuned. These are tuned for each algorithm $k \in \mathcal{L}$ on each task $j \in \Theta$. We take the view of Agarwal et al. [23], that focus should turn to the reliable measurement of algorithm performances *after* hyperparameter tuning. This is since algorithm performance levels can suffer depending on the choice of hyperparameters. Hyperparameters should be tuned to their best possible performance using methodologies such as grid searches or informed choices. Recall from section 2.2 that statistical claims about algorithm performances pertain only to the algorithm with its hyperparameter search method (maximally) and (minimally) to the algorithm with the specific set of selected hyperparameters. In our view, careful selection of network architectures, activation functions, numerical optimisers and code-level implementation differences [22, 100] can be considered, abstractly, as aspects of hyperparameter optimisation. It is therefore important to not only report the hyperparameters selected but also to publish the algorithm code used as part of the reporting of experimental details and evaluation procedures [1].

3.1.2.4 Runs Per Task

Having appropriately tuned hyperparameters, we have algorithms ready to be trained and tested. As it is important to have sufficient data to make statistically robust claims, these algorithms will be used on each task for approximately 5-10 runs per task. This is since Agarwal et al. [23] find that stratified aggregation (across tasks) and uncertainty measurements can be suitably calculated for sample sizes of only a *handful of runs* per task. We denote each run per task as indexed by $n \in \{1, ..., N_j\}, N_j \in \mathbb{N}$. The low number of runs recommended is in order to comply with the *usability* and computational tractability requirements mentioned explicitly in Jordan et al. [21] and discussed at length above. We also recommend against the cherry-picking of some top N runs for reporting [21].

3.1.2.5 Duration of Runs (Training Curves)

Following from above, we have M tasks and N_j runs per task j. Each of the algorithms $k \in \mathcal{L}$ will be trained for a certain number of timesteps, T, which will be plotted to produce a training curve. We note that is is most common in MARL for algorithms to be trained for $T = 2$ million timesteps [1] although we do not fixate on this number and note that it is up to the evaluator to decide on the length of training, T. An important consideration is raised by Papoudakis et al. [88] who note that as off-policy algorithms have improved sample efficiency as compared to on-policy algorithms, the on-policy algorithms need be trained for a factor of 10 more timesteps for fair comparison.[88] This recommendation is reiterated in Gorsane et al. [1]. Another consideration is to look at the typical times for similar algorithms in the literature to empirically converge to optimal performances.

3.1.2.6 Frequency of Evaluating Data-Points across the Training Curves

Following the recommendation by Colas et al. [19], we recommend that a measure of an algorithm's performance should be taken after every $t^* \in \mathbb{N}$ timesteps across a training curve. This measurement should be taken as the mean of $E \in \mathbb{N}$ evaluation episodes. The E evaluation episodes are conducted independently from training, using the roll-outs of the fixed, check-pointed policy parameters at times t. We note that it is up to the evaluator to decide on how frequently to evaluate the fixed policy roll-out averages (the length of the interval t^*) and also to decide on the number of evaluation episodes E to average over at every evaluation interval. Interestingly, Gorsane et al. [1] find that it is most common in the cooperative MARL research field for $E = 32$ evaluation episodes to be averaged over at every $t^* = 10000$ timesteps.

3.1.2.7 At Which Timestep to Take Performance Point Estimates

We recommend using the *final metric* [92] as a point estimate for performance of each algorithm on each learning curve produced. This is the average of the E evaluation episodes of the final check-pointed policy on the learning curve at time T, and is used by Machado et al. [14], Henderson et al. [17], Chan et al. [20] and Agarwal et al. [23]. This contrasts with the use of the *absolute metric* as is recommended by Gorsane et al. [1] and Colas et al. [92] which is calculated using the average performance (over E evaluation episodes) of the best check-pointed policy across the learning curve. Importantly Agarwal et al. [23] suggest that using the absolute metric results in positive bias as noted in section 2.2.3.

Hence we run our chosen $|\mathcal{L}|$ algorithms with tuned hyperparameters on each of the M tasks for $N_j \in [5, 10], N_j \in \mathbb{N}$ runs per task [23]. The algorithms are evaluated every t^* timesteps for E evaluation episodes and the training process is terminated after T timesteps. At T timesteps we take the final metric as our performance measure, taking the mean of the E evaluation episodes at timestep T. This leaves us with $M \times N_j$ data points for each algorithm corresponding to the M tasks and N_j runs per task. We denote the final metric as $x_{n,j}(k)$.

It is at this point that we can plot the sample efficiency curves which are plotted for each algorithm on each task[89]. The reliability metrics by Chan et al. [20] can also be used here. These are particularly useful for measuring uncertainty of the per-task algorithm performances.

Given our data, recall that the second broad step in the MCDA process is model building and preference

[88] Papoudakis et al. [88] note that wall-clock times of on-policy algorithms, that are trained for a factor of 10 more timesteps than similar off-policy algorithms, are almost equivalent to the wall-clock times of similar off-policy algorithms.

[89] It is harmless to plot these by taking the min-max normalisation as proposed by Bellemare et al. [27]. This is since we are not aggregating across tasks so consistent performance scales are not required and it is sometimes more convenient to see scores on the interval $[0, 1]$.

modelling. Preference modelling, in turn, consists of two primary components. These are, firstly, the selection of a preference model for each individual criterion and, secondly, the choice of an aggregation model. Next we discuss the construction of partial value functions for capturing the decision-maker's preferences for different levels of reward on each task.

3.1.3 Preference Modelling and Model Building

We use MCDA value function methods for constructing partial value functions for each criterion. The partial value functions map performance scores on the RL tasks to a decision-maker's preferences. The construction of such value functions thus accounts for the non-linear scaling of task difficulty with performance and, hence, the non-linear scaling of a decision-maker's preferences with performance. These partial value functions are expressed as functions g_j for tasks $j \in \Theta$ that take performance scores $x_j(k)$, for algorithms $k \in \mathcal{L}$ and provide a map $g_j : x_j(k) \longrightarrow \mathbb{R}$.

We recommend that the partial value functions be constructed by domain experts who have advanced knowledge of the RL tasks being used. This should involve knowledge of the tasks' reward functions and knowledge of how algorithm behaviours on the tasks correspond to different levels of reward. Constructing partial value functions in tandem with such experts, using either of the (below) suggested methods of preference elicitation, performing consistency checks about the experts' preferences and doing so iteratively results in a more robust construction [194, 195].[90]

The partial value functions need be complete and transitive in terms of a user's preferences for different algorithms [25, 195]. Hence preferences for algorithms will have a complete weak ordering. Constructing the functions on the interval scale yields an additive value function model when aggregating across preferentially independent and measurable criteria, this is expressed in Equation 25.

We start our construction of our partial value functions by declaring two reference points per task. These can either reflect the local performance maxima and minima of the performance scores (for each task) in terms of the scores collected in the data collection process, or they can reflect the global possible minima and global possible maxima for each of the tasks, if available. The local minimum and maximum for a task $j \in \Theta$ can be denoted as $\min_{k \in \mathcal{L}} x_j(k)$ and, similarly, $\max_{k \in \mathcal{L}} x_j(k)$. The global minimum and maximum for a task j correspond to the lower and upper bounds, respectively of the task's reward function, $r(s, a, s')$. The selection of either the local or global reference points should be consistent across partial value function construction however, there is no fundamental reason for selection of either [195].

Once the reference points are selected we define our partial value functions to have minimum values of 0 and maximums of 100, such that $g_j : x_j(k) \longrightarrow [0, 100]$. We note that our value functions will be monotonically increasing since our preference is for an increase in average reward, and therefore an increase in the average performance score. This implies that $g_j(x_{min}(k)) = 0$ and $g_j(x_{max}(k)) = 100, \forall j \in \Theta, k \in \mathcal{L}$.

3.1.3.1 Constructing Partial Value Functions

In this dissertation we make use of the difference method by von Winterfeldt and Edwards [192] for eliciting the decision-maker's preferences when constructing the partial value functions.[91] This method was discussed in section 2.4.4.2 and constructs partial value functions for each task $j \in \Theta$ by first defining a unit level of value u. This represents the value of increasing from between one fifth to one tenth of the range of

[90] In the presence of multiple experts, performing consistency checks across the experts' opinions may be useful in workshopping a consensus opinion for the final form of the partial value functions [194].

[91] Either of the other two indirect methods mentioned in section 2.4.4.2 for constructing partial value functions are equally useful. Moreover, the evaluator can use one of the other methods to perform consistency checks to protect against judgemental errors when constructing the partial value functions.

the reward function. We choose the value $\kappa \in [\frac{1}{10}, \frac{1}{5}]$ such that we have our desired unit level of value, defined to be $u = g_j(a_{1,\max}) - g_j(x_{min})$. This has that $a_{1,\max}$ is defined to be the upper bound of the range we selected for our unit level of value $a_{1,\max} = \kappa(x_{max} - x_{min})$. This range defines the interval $A_1 = [x_{min}, a_{1,\max}]$. Our task is then to construct the partial value function by finding the subsequent upper bound of an interval A_2. The interval A_2 is defined such that the partial value of an increase over its range is equal to our defined unit level of value u. Here, A_2 has a lower bound equal to the upper bound of the previous interval $a_{2,\min} = a_{1,\max}$ such that $A_2 = [a_{1,\max}, a_{2,\max}]$. This means we must find $a_{2,\max}$ such that $g_j(a_{2,\max}) - g_j(a_{1,\max}) = u$.

We find $a_{2,\max}$ by asking the decision-maker to name the amount of reward for which an increase in value from x_{min} to $a_{1,\max}$ is equivalent to an increase in value from $a_{1,\max}$ to $a_{2,\max}$. Given A_1 and A_2 we then seek $A_3 = [a_{2,\max}, a_{3,\max}]$. In general we find $a_{\alpha,\max}$, with $\alpha \in I, I = \{1, 2, ..., \zeta\}$, by asking the decision-maker to name the amount of reward for which an increase in value from known lower bound $a_{\alpha-1,\max}$ to unknown upper bound $a_{\alpha,\max}$ is equivalent to u: i.e. $g_j(a_{\alpha,\max}) - g_j(a_{\alpha-1,\max}) = u$. This process is continued until $a_{\zeta,\max} = x_{max}$ for some finite $\zeta \in \mathbb{N}$.

Finding points $a_{\alpha,\max}$ for $\alpha \in I$ establishes intervals A_α such that $\bigcup_{\alpha \in I} A_\alpha = [x_{min}, x_{max}]$. Using defined values for x_{min} and x_{max} and given a monotonic increasing partial value function with $g_j(x_{min}(k)) = 0$ and $g_j(x_{max}(k)) = 100$ then, we can use the $a_{\alpha,\max}$ to find ordered pairs $(a_{\alpha,\max}, g_j(a_{\alpha,\max}))$ that lie on our partial value function. These are then used to construct a piece-wise linear partial value function. The values of $g_j(a_{\alpha,\max})$ depend on the number of intervals ζ that we find in our construction process. For instance, in the case that $\zeta = \zeta^*$ then $g_j(a_{\alpha,\max}) = \frac{100\alpha}{\zeta^*}$. The provision of as few as five points is sufficient for encapsulating the user's preferences in a partial value function according to various simulation studies [195, 197, 198]. Putting this together, we construct the piece-wise linear partial value functions for each task $j \in \Theta$ to be:[92] [93]

$$
\begin{aligned}
g_j(x) &= \begin{cases}
0 & \text{if } x \in (-\infty, x_{min}) \\
g_j(x_{min}) + \frac{g_j(a_1) - g_j(x_{min})}{a_1 - x_{min}}(x - x_{min}) & \text{if } x \in [x_{min}, a_1) \\
g_j(a_\alpha) + \frac{g_j(a_{\alpha+1}) - g_j(a_\alpha)}{a_{\alpha+1} - a_\alpha}(x - a_\alpha) & \text{if } x \in [a_\alpha, a_{\alpha+1}), \alpha \in \{1..., \zeta - 2\} \\
g_j(a_{\zeta-1}) + \frac{g_j(x_{max}) - g_j(a_{\zeta-1})}{x_{max} - a_{\zeta-1}}(x - a_{\zeta-1}) & \text{if } x \in [a_{\zeta-1}, x_{max}] \\
100 & \text{if } x \in (x_{max}, \infty)
\end{cases} \\
&= \begin{cases}
0 & \text{if } x \in (-\infty, x_{min}) \\
\frac{\frac{100}{\zeta}}{a_1 - x_{min}}(x - x_{min}) & \text{if } x \in [x_{min}, a_1) \\
\frac{100\alpha}{\zeta} + \frac{\frac{100}{\zeta}}{a_{\alpha+1} - a_\alpha}(x - a_\alpha) & \text{if } x \in [a_\alpha, a_{\alpha+1}), \alpha \in \{1..., \zeta - 2\} \\
\frac{100(\zeta-1)}{\zeta} + \frac{\frac{100}{\zeta}}{x_{max} - a_{\zeta-1}}(x - a_{\zeta-1}) & \text{if } x \in [a_{\zeta-1}, x_{max}] \\
100 & \text{if } x \in (x_{max}, \infty)
\end{cases}
\end{aligned}
\tag{29}
$$

Similarly, in this dissertation we also make use of the *bisection method* [192], as described in section 2.4.4.1. Here we ask the decision-maker to find $x = x_{50}$ for each task $j \in \Theta$ such that:

$$
g_j(x_{max}) - g_j(x_{50}) = g_j(x_{50}) - g_j(x_{min}) \iff 100 - g_j(x_{50}) = g_j(x_{50}) \tag{30}
$$

In other words we find the decision-maker's mid-point (x_{50}) in terms of preference between the worst (x_{min}) and best (x_{max}) possible performance levels. This will then have that $g_j(x_{50}) = 50 \; \forall j \in \Theta$.

Thereafter we ask of the decision-maker to identify the midpoint (x_{25}) in terms of preference between the

[92] This uses the equation of a straight line from two points (x_1, y_1) and (x_2, y_2) as $y = y_1 + \frac{\Delta y}{\Delta x}(x - x_1)$.

[93] We drop the subscript max for simplicity such that $a_{\alpha,\max}$ is denoted as a_α.

worst possible performance level x_{min} and the performance level at x_{50} such that:

$$g_j(x_{50}) - g_j(x_{25}) = g_j(x_{25}) - g_j(x_{min}) \tag{31}$$

Similarly, we find x_{75} such that:

$$g_j(x_{max}) - g_j(x_{75}) = g_j(x_{75}) - g_j(x_{50}) \tag{32}$$

This gives $g_j(x_{25}) = 25$ and $g_j(x_{75}) = 75$ $\forall j \in \Theta$. One may continue in this fashion until sufficient complexity is accounted for in terms of the decision-maker's preferences. However, as mentioned previously, the provision of as few as five points is sufficient according to several simulation studies [195, 197, 198].

Given an *ordered* set of selected performance score levels $A_{bisect} = (x_{min}, ..., x_{25}, ..., x_{50}, ..., x_{75}, ..., x_{max})$ elicited from the decision-maker using the bisection method, one can translate these into interval endpoints $a_\alpha, \alpha \in I, I = \{0, ..., \zeta\}$ in order to construct a piece-wise partial value function using Equation 29. Once again we have that $x_{max} = a_{\alpha=\zeta}$ and $x_{min} = a_{\alpha=0}$ such that $A_{bisect} = (a_0, a_1, ..., a_{\zeta-1}, a_\zeta)$ with $\zeta = |A_{bisect}| - 1$. Using these values we use Equation 29 to construct our partial value function approximations for all $j \in \Theta$.

3.1.4 Data Aggregation

Given partial value functions which provide performance scores on an interval scale for preferentially independent criteria, we can make use of an additive value function model for aggregating across tasks [195]. This model, taken from section 2.4.5, is found below in Equation 33. It results in one performance score, y, for any alternative $k \in \mathcal{L}$ and does so by aggregating across the partial value functions of all tasks $j \in \Theta$ and runs $n \in \{1, ..., N_j\}$ such that:

$$y(x_{j_1}(k), ..., x_{j_M}(k)) = \sum_{j \in \Theta} \sum_{n=1}^{N_j} q_{j,n} \left[g_{j,n}(x_{j,n}(k)) \right] = \sum_{j \in \Theta} q_j \sum_{n=1}^{N_j} q_{n|j} \left[g_{j,n}(x_{j,n}(k)) \right] \tag{33}$$

Criteria in MCDA can be expressed in a tree structure called a value tree. Equation 33 reflects the value tree. In our case the root node represents the value function $y(k)$ which is comprised by summing up the (weighted) values of the upper level criteria, corresponding to tasks $j \in \Theta$. The value of each task is comprised by summing up the (weighted) values of each of the runs $n \in \{1, ..., N_j\}$ per task. The value tree is represented by our diagram in Figure 13.

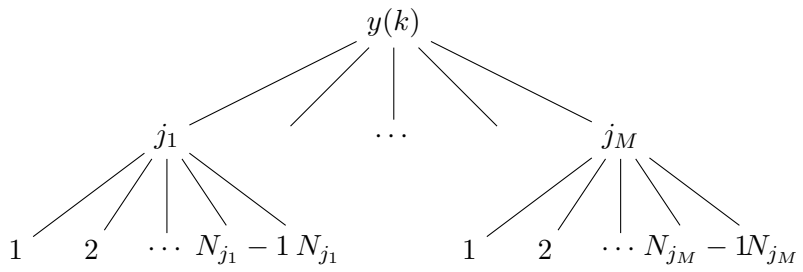

Figure 13: Value tree representing the components that comprise the value function $y(k)$ in Equation 33. The function is comprised by summing over the (weighted) performance scores of each task $j_1, ..., j_M$. These in turn are comprised by summing the (weighted) performance scores of each run per task $1, ..., N_j$.

The weights in Equation 33 can be expressed either as $q_{j,n}$ which are the cumulative weights of each leaf node in the value tree where $q_{j,n} = q_j q_{n|j}$ or they can be expressed as q_j and $q_{n|j}$ which are the relative weights of one node with respect to its siblings (normalised to sum to 1). The relative weights are assessed within families, sharing the same parent nodes. In our case we begin, at the lowest level of our value tree by assessing the weights within the families of each parent task node j. These correspond to the lower level criteria, which we weight equally as they correspond to runs on the same task and are therefore measured on the same reward scale and are of the same relative importance. This gives $q_{n|j} = \frac{1}{N}$ which yields:

$$y(x_{j_1}(k), ..., x_{j_M}(k)) = \sum_{j \in \Theta} \frac{q_j}{N_j} \sum_{n=1}^{N_j} [g_j(x_{j,n}(k))] \tag{34}$$

We then assess the relative weights q_j within the next family, under the same parent node, the root node. The relative weights are reflected by the tree in Figure 14. In order to account for both the concept of the relative importance of each task with respect to each other and the extent to which the reward scales used discriminate between tasks, we make use of the swing weight method for selecting weights $q_j, \forall j \in \Theta$.

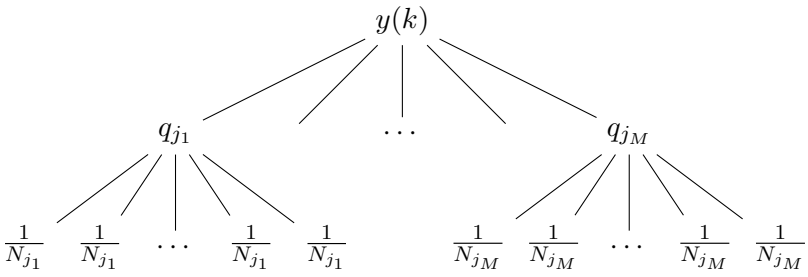

Figure 14: Value tree representing the weights of each component that comprises the value function $y(k)$ in Equation 34. The weights q_j, $j \in \Theta$ refer to the relative weight of each task with respect to the other tasks. The leaf node relative weights are $q_{n|j} = \frac{1}{N_j}$. These are, therefore, equal within families as they represent the relative weightings of each run $n \in \{1, ..., N_j\}$ for each task j with respect to all other runs for that particular task.

Given M tasks we ask of the decision-maker to imagine an algorithm which scores at its worst for all tasks $x_j = x_{j,min} \forall j \in \Theta$. This gives the ordered tuple $(x_{j=1,min}, ..., x_{j=M,min})$. The decision-maker must then select from M options which tuple they prefer the most when all tasks perform at their minima except for one task j^* which has a swing in performance and performs at its maximum with $x_{j=j^*} = x_{j=j^*,max}$. The task for which the decision-maker prefers a *swing* from worst to best in, is the task which receives the largest weight q_j.

The decision-maker must then consider choosing from $M-1$ options where an alternative performs at its worst in all tasks except for the task j^* and some second task $j_2^* \in \{\Theta - j^*\}$. The task j_2^* for which the decision-maker prefers a swing in performance from worst to best in, is the task which receives the second largest weight. Continuing in this fashion, a rank ordering of weights is established.

Values are then assigned to weights by asking the decision-maker to assess the relative importance of swings in performance (from worst to best) of all tasks with respect to the task with the largest weight. The largest weight is assigned a (standardised) value of 100 such that $q_{j^*} = 100$. Then we ask: *What is the relative value of a swing from worst to best for the other tasks as compared to a value of 100 for the highest ranked task?* All weights can be assigned values based on this assessment

This process requires $M-1$ comparisons per family of M tasks. This approach offers the most tractable assignment of weights according to Scott [25]. A minimal number of consistency checks must then be conducted, as proposed by Stewart [195].[94] This is done by specifying a second reference task and having

[94]Performing consistency checks and assessing the sensitivity of performance evaluations to changes is weights is recom-

all other tasks cross-checked against this. Lastly, all relative weights within families are normalised to sum to 1.

Combining the weights with the partial value functions yields all the components necessary to assess the value y for each algorithm $k \in \mathcal{L}$. These can then be compared to resolve our decision problem.

3.1.5 Other Details to Report

As mentioned previously, the last component of an effective performance evaluation is to provide a report of all hyperparameters used, the code used, open-source data, experimental details, hardware, algorithm training wall-clock times and evaluation details. These allow for reproducible and replicable performance assessments [1, 17, 22, 100].

Other metrics which can enhance the performance assessment include the Mann-Whitney U statistic representing the *probability of improvement* when using one algorithm instead of another [23]. This can be found in Equation 22. This does not suffer from the aforementioned normalisation or biased weighting problems but does not take into account absolute magnitudes of algorithm scores. Adapting the probability of improvement by using a set of human scores as one set of scores yields the *superhuman probability* metric [23] which may be useful in some contexts.

Finally it is important to plot sample efficiency curves to depict performance across training runs [1, 19, 20, 23].

We now refer the reader to the summarised results of the above work. This summary can be found in section 3.2 and comprises our guideline for improved evaluation. Thereafter the reader is invited to view a discussion of this work in section 3.3 before continuing to section 4 for a proof-of-concept test of the guideline using our own collected RL algorithm performance data.

mended to ensure the robustness of the choice of weights [195]. Furthermore, using the specifically designed swing weight elicitation procedure and combining this with expert knowledge adds to the robust choice of weights.

3.2 Results: A Guideline for Deep RL Performance Evaluations

Putting together the work outlined in section 3.1, we are able to produce a coherent guideline for deep RL empirical algorithm performance evaluations. This is summarised below. Thereafter, section 3.3 will discuss this guideline.

The guideline produced provides an answer to our third research question: *Can MCDA be used to benefit a guideline for coherent empirical algorithm performance evaluation criteria, and if so, how?* The production of the guideline below demonstrates an effective use of MCDA at providing a coherent framework for the evaluation of algorithms across several tasks. This includes a coherent aggregation process that takes into account the different reward scales of each task and the relative importance of each task with respect to one another. The MCDA framework has enabled us to address the two problems with evaluation as identified in the literature: flawed normalisation procedures and biased weightings of tasks. These are resolved using processes advised by the MCDA framework, namely value function methods and the swing weight method respectively. The value function methods allow us to re-scale average rewards achieved by various algorithms whilst taking into account the non-linear scaling of the reward function with the task difficulty. The swing weight method allows us to account for the different reward scales between tasks and the relative importance of the tasks in order to aggregate across all the tasks. Thus, integrating MCDA with other recommendations for evaluation found in the literature allows us to construct the below guideline for performance evaluations of deep RL algorithms.

A Guideline for Coherent Deep RL Empirical Algorithm Performance Evaluations

- Select algorithms $k \in \mathcal{L}$ to evaluate.
- Select tasks $j \in \Theta$ for algorithm evaluation. These must be a complete set of non-redundant, measurable and preferentially independent tasks.
- Tune the hyperparameters of each $k \in \mathcal{L}$ on each $j \in \Theta$ as best as possible [13]. This includes a careful selection of code-level optimisations such as network architectures, numerical optimisers and activation functions [22, 100].
- Select the number of runs $N_j \in \mathbb{N}$ per task j. $5-10$ runs are recommended by Agarwal et al. [23].
- Select the duration of the runs, the number of timesteps T that each algorithm will run for. Papoudakis et al. [88] suggest training on-policy algorithms for a factor of 10 more time-steps than off-policy algorithms for fair comparison. Gorsane et al. [1] recommend $T = 2$ million due to commonality across MARL papers.
- Select the frequency of evaluating data-points across the runs by selecting the number of timesteps $t^* \in \mathbb{N}$ whereby after every t^* timesteps on the training curve, a performance evaluation will take place [19]. Gorsane et al. [1] recommend $t^* = 10000$ due to commonality across MARL research.
- Select the number of evaluation episodes $E \in \mathbb{N}$, whereby every t^* timesteps E evaluation episodes will be completed using a check-pointed algorithm policy at that particular timestep. Gorsane et al. [1] recommend $E = 32$.
- Run algorithms $k \in \mathcal{L}$ on each of the $j \in \Theta$ tasks for N_j runs per task.
- Whilst conducting the runs, evaluate the algorithms every t^* timesteps. Here, take the check-pointed policies every t^* timesteps and calculate the mean reward of E evaluation episodes for these.
- End the runs after T timesteps and take the mean reward of the final check-pointed policy as the performance metric for a particular algorithm k on run n, on task j [14, 17, 20, 23, 92]. We denote this *final metric* as a performance score x.
- Plot the training curves for each algorithm k on each task j.
- Consider using the reliability metrics by Chan et al. [20] to document uncertainty.
- Construct interval scale partial value functions $g_j : x_{n,j}(k) \longrightarrow [0, 100]$ for each task $j \in \Theta$. These will provide a complete weak ordering of algorithms in terms of a user's preferences for different levels of reward on each task j.
- Construct the $g_j(x)$ by first identifying either the local or global minimum and maximum possible scores on each of the tasks x_{min} and x_{max}. Assign these values of 0 and 100 such that $g_j(x_{min}(k)) = 0$ and $g_j(x_{max}(k)) = 100$.
- Construct a piece-wise linear partial value function using the elicitation methods by von Winterfeldt and Edwards [192], found in section 3.1.3.1.
- Assign weights to each of the tasks using the swing weight method, seen in section 3.1.4.
- Equally weight the runs per task using $\frac{1}{N_j}$ as the weights.
- Provide an additive aggregate empirical performance score of the algorithms' performances across tasks and runs by using the aggregate aggregation model that combines the constructed partial value functions and above weights: $y(x_{j_1}(k), ..., x_{j_M}(k)) = \sum_{j \in \Theta} \frac{q_j}{N_j} \sum_{n=1}^{N_j} [g_j(x_{j,n}(k))]$.
- Compare aggregate performance scores to assist in choosing a preferred algorithm.
- Report and provide code, data, hyperparameters, hardware, algorithm wall-clock times, experimental procedures and evaluation details.
- Consider reporting the Mann-Whitney U probability of improvement and *superhuman probability* [23].

3.3 Discussion: A Guideline for Deep RL Performance Evaluations

Our guideline, found in section 3.2, directly addresses the main goal of this dissertation: to address deep RL empirical algorithm performance evaluations. Stemming from the lack of rigour in this field, identified through answering our second research question, we were motivated to rectify the problems identified with RL evaluation. The guideline results from insights drawn from an extensive literature review in RL and the use of the MCDA framework.

The use of the MCDA framework provides an answer to our third research question, which is the focus of this discussion: *Can MCDA be used to benefit a guideline of coherent deep RL empirical algorithm performance evaluations criteria, and if so, how?* The integration of MCDA into the guideline for evaluation demonstrates that it *can* be used to benefit such a guideline. The benefit stems from MCDA addressing two problems with evaluation which we identified in our literature review.

The first of these problems was the use of the min-max normalisation function $g(x, j) = \frac{x - \min_k x_{k,j}}{\max_k x_{k,j} - \min_k x_{k,j}}$ [21, 195]. This function is widely used, as reflected by commonly cited works like Bellemare et al. [27] and as reflected by our new work in Gorsane et al. [1]. However, criticism points toward this function being sensitive to the reward scales used [21]. This leads to aggregation being incompatible across tasks due to performance scores clustering in different regions of $[0, 1]$, as seen in our example in section 2.2.3.2. Another particular problem with the min-max normalisation function is that it requires a linear scaling of performance scores with task difficulty [21]. This is often not the case for many RL tasks.

The min-max normalisation is specifically mentioned in the MCDA literature due to its widespread misuse [195]. MCDA requires transformations of performance scores x using functions $g(x)$ that abide by the interval scale property, as illustrated in section 2.4.4. However, the min-max transformation does not in general represent an interval scale of preferences since there is often a changing rate of trade-off (in terms of preferential value) between tasks as performance scores x vary across their range [21, 195].

Several simulation studies provide further indictment of the min-max normalisation [195, 197, 198]. These state that using such inappropriate linear transformations leads to biased results due to the favouring of extreme outcomes, such as algorithms that score very well on a few tasks but very poorly on others, rather than algorithms that have more balanced results. Moreover, Stewart [195] argues that the largest source of bias in reported results is the use of inappropriate additive value function models. However, these studies [195, 197, 198], demonstrate that the use of *piecewise linear approximations* to partial value functions (Equation 29) largely eliminates the bias.

The second problem identified with evaluation, which MCDA addresses, is the uniform weighting of tasks when providing an aggregate performance measurement across tasks [21]. As discussed in section 2.2.3.2, this can introduce bias. As we want to conform to the opinions of Whiteson et al. [12], Jordan et al. [21] and Agarwal et al. [23], to use multiple tasks for evaluation, our guideline made use of the swing-weight method to mitigate this bias. This allows our additive value functions (Equation 34) to account for both the relative importance of each task with respect to each other and to account for the extent to which the reward scales used discriminate between the tasks [195].

Hence, our guideline uses partial value functions $g(x)$ and the swing weight method (see sections 3.1.3.1 and 3.1.4) to address the problems of non-linear scaling and biased weightings which were identified (through the answering of our first research question) as problems that still remain with RL evaluation. Despite this, a few shortcomings with MCDA remain.

Firstly, our additive value functions $y(\mathbf{x})$[95] are never completely and explicitly known [197]. This is due to the decision-maker's preferences never being fully represented by the set of criteria and due to our inability to guarantee the achievement of this goal.[96] Hence, we can only identify a subset of potential criteria that will

[95] Here, we denote the vector of performance scores $(x_{j_1}(k), ..., x_{j_M}(k))$ as $\mathbf{x}$ for shorthand.

[96] This relates to the aphorism by Box and Draper [201]: 'All models are wrong, but some are useful'.

represent the decision-maker's preferences *with high likelihood* and that will yield a performance evaluation that will adequately compare alternatives *with high likelihood* [197]. Selection of criteria (tasks) from this subset will therefore result from the decision-maker's holistic judgement which allows for consideration of non-modelled aspects of the decision problem. This comment relates to our guideline's recommendation to attempt to find a complete and non-redundant set of tasks for evaluation. As noted, this is a non-trivial challenge, and finding such a set is hard to quantify. In this dissertation, we have not recommended any particular solution to this problem and have instead trivially recommended selecting several tasks which test the algorithms for their intended use. This provides a compromise that agrees with the literature [1, 12, 16, 21, 23, 74]. Moreover, we warn against the cherry-picking of specific subsets of tasks that benefit one's desired algorithm's performance (as discussed in section 2.3).

A second shortcoming with MCDA is the involvement of uncertainty and subjectivity in the construction of value functions and the elicitation of swing weights. The true forms of the partial value functions $g(x)$ will not generally be self-evident at the beginning of evaluations [198]. Moreover, these true forms might never be identified. However, Stewart [198] notes that the construction of partial value functions can be less susceptible to judgemental errors when at least five reference points are used for construction and when imprecisions in inputs are not further than 10% away from true values. The use of elicitation methods like the bisection and difference methods by von Winterfeldt and Edwards [192] helps to identify *indifference* points in terms of decision-maker preferences on the task reward scales. These methods offer improved robustness as compared to direct methods of construction [194, 195].[97] [98]

The construction of partial value functions necessarily requires familiarity with the RL tasks as well as the behaviour of RL algorithms on the tasks. Specific *expert* knowledge about the reward functions of these tasks is required. Therefore, it is useful to recommend the construction of such value functions by the authors of RL tasks. This is because the level of knowledge required for accurate construction of partial value functions may be difficult to obtain for any individual.[99] We recommend that the authors of RL tasks should publish details specifically about their task's reward functions $r(s, a, s')$ so as to provide a maximal amount of information for the construction of partial value functions $g(x)$ in their absence. In the interest of furthering robustness, we recommend an iterative process of questioning the experts, where consistency checks are conducted in order to ensure the accuracy of value judgements [194]. Moreover, in the presence of more than one expert, a consensus opinion on which partial value function to use can be developed.

The elicitation of weights is less susceptible to uncertainty and subjectivity. The consistency checks for weights required by Stewart [195] ensure a relative level of robustness (see section 3.1.4). Furthermore, one could run a Monte Carlo simulation where weights can be drawn from distributions of possible values to establish decision boundaries for the weights [195]. Alternatively, sensitivity analyses can be conducted where the values of the weights are varied slightly to determine if other algorithms are potentially optimal. The use of such sensitivity checks is essential in order to ensure the *robust* identification of a preference ordering for the algorithms we are evaluating [198].

Crucially, the use of MCDA means that one algorithm cannot be determined the *winner* in a competitive test [24] if small perturbations in weights lead to a change in its ranking amongst other algorithms [198]. MCDA thus seeks to inform the evaluator and not to (in general) form a precise ordering of alternatives. An iterative process, whereby the additive value function models are reassessed, is therefore recommended [195]. In the case that an evaluation has taken place where the decision-maker deems the results to be unsatisfactory, the model can be reassessed in conjunction with the evaluator to see if criteria need be added or weights need be adjusted.

Having discussed the strengths and weaknesses of the use of MCDA, we now compare our guideline with

[97] See sections 2.4.4 and 3.1.3.1 for discussion on direct and indirect methods of construction of partial value functions.

[98] The discussed indirect methods are equivalent to the recommended elicitation methods used by analysts for constructing utility functions in the field of subjective expected utility theory and to some recommended methods for eliciting Bayesian priors from domain experts [194].

[99] This may be particularly true for multi-agent tasks.

that provided by our published work in Gorsane et al. [1]. A first point of difference is that Gorsane et al. [1] apply their guideline to cooperative MARL algorithms only where we claim ours to be (more broadly) applicable to both single-agent RL (SARL) and cooperative MARL algorithms. In our view, the guideline by Gorsane et al. [1] should be considered applicable to SARL algorithms due the problem equivalence between SARL and cooperative MARL algorithm evaluations (discussed in section 2.3). This is, therefore, a minor difference.

More significantly, a second difference between the two guidelines is that Gorsane et al. [1] uses the *absolute metric* for algorithm performance measurements $x_{j,n}$ for a particular run n and task j. This is defined by Colas et al. [92] to be the average algorithm performance (across E evaluation runs) of the *best* check-pointed policy found in training. Our guideline instead recommends using the *final metric* [92] which is the average of E evaluation episodes of the last check-pointed policy at the *end* of training, and as is used by Agarwal et al. [23]. This avoids the positive bias that results from selecting the best check-pointed policy [23].[100]

A third difference between our work and the work of Gorsane et al. [1] is their use of the min-max normalisation function for aggregation across tasks, where instead we make use of MCDA. Consequently, Gorsane et al. [1] make use of the recommendations by Agarwal et al. [23] for using the inter-quartile mean (IQM) (see Equation 23) for stratified aggregation. The stratified aggregation using IQM imposes a biased weighting of tasks during aggregation [21], similarly to using a mean. This problem we avoid by using the swing weight method. Furthermore, the IQM excludes sampled scores from aggregation, only on the basis of which quartiles these scores fall in (relative to other scores). The scores are ordered based on their min-max normalised transformed values. This means that their exclusion (using the IQM) is based on a comparison of values that are not necessarily compatible (due to the problems with the min-max normalisation function).

Despite these issues, and due to their use of stratified aggregation, Gorsane et al. [1] are able to include a second recommendation of Agarwal et al. [23]: The use of *stratified bootstrap confidence intervals* for measuring uncertainty. Contrasting this with our guideline identifies another shortcoming, that the use of MCDA does not provide a clear way to measure uncertainty of the aggregated algorithm performance scores $y(k)$. We are, however, able to use the reliability metrics by Chan et al. [20] to measure uncertainty in algorithm performance on a *per task* basis.

We are also unable to use the performance profiles (see Equation 24 and Figure 6) as are recommended by Agarwal et al. [23]. This is because these too rely on stratified aggregation. Agarwal et al. [23] choose stratified aggregation as a technique to help resolve the issues caused by small sample sizes (see Figure 4) which are prevalent amongst the computationally expensive RL research field [21, 23]. In the case that our computational capacity to do RL improves in the future, the recommendations to use the *performance percentiles* by Jordan et al. [21] (see section 2.2.3.2) may apply. These offer an alternative normalisation method, with a corresponding method to measure uncertainty, but rely on large sample sizes.

Other considerations of Jordan et al. [21] include the requirements for an evaluation procedure to be *scientific, computationally tractable, usable and non-exploitative*, as is discussed in section 2.2.3. Our guideline (and the use of MCDA) is compatible with the computational tractability requirement as we focus on the recommendation by Agarwal et al. [23] to provide an evaluation procedure that can be used with only a *handful of runs*. Furthermore, we focus on the usability and non-exploitative requirements by recommending the use of multiple tasks and by recommending an aggregation procedure that takes into account the extent to which the different reward scales used discriminate between tasks. However, our guideline falls short on the recommendations to quantify uncertainty and quantify hyperparameter tuning times as part of our algorithm comparisons.

Lastly, the requirement to be scientific demands that the evaluation procedure answer specific research questions and test hypotheses [21]. While our guideline may appear to be only compatible with the competitive testing [24] of algorithms as opposed to their scientific testing, the techniques developed here can

[100] See section 2.2.3.1.

be used to evaluate various algorithmic choices. Therefore, they may form an important component of careful experimentation, as part of a scientific test.

In the next section, we subject our evaluation guideline to deep RL algorithm performance data and provide a proof-of-concept test to assess its use. We examine this in light of our fourth and final research question on the effectiveness of the use of our RL guideline.

4 Testing our Guideline for Evaluation

In this section we aim to respond to our fourth and final research question which asks about the effectiveness of using our proposed guideline for deep RL performance evaluations. Here, we provide a proof-of-concept test of our guideline[101] using deep RL algorithm performance data. Accordingly, subsection 4.1 outlines the use of our guideline. Our choices of algorithms and RL tasks are described along with the implementations used. The results are found in subsection 4.2 and our guideline's use and efficacy is discussed in subsection 4.3.

4.1 Method: Using Our Guideline for Evaluating Deep RL Algorithm Performances

This subsection provides the methodology for running an evaluation of the empirical performances of three deep RL algorithms. The subsection aims to demonstrate how to use the guideline constructed in section 3 by providing a proof-of-concept test using deep RL algorithm performance data. The subsection will describe our use of the guideline by running through its recommendations in sequential order. Specifically, we will discuss the tasks selected for evaluation, the data collection process and, finally, the data aggregation process using multi-criteria decision analysis (MCDA). We aim to answer our fourth, and final, research question on the effectiveness of using our guideline. The guideline's hypothesised efficacy will be tested through this proof-of-concept experiment.

4.1.1 Algorithm Selection

We begin by selecting the algorithms that we want to evaluate and compare. We choose the following three deep RL algorithms for evaluation[102]:

- **Deep Q Networks (DQN)** [8, 35]:

  This an off-policy value function algorithm that uses Q-learning as seen in Equation 4. DQN learns value functions, Q_{π_θ}, by making use of deep neural networks as function approximators and randomised replay buffers for storing the agent's experienced trajectories of states, actions and rewards.[103] An example policy using DQN's learnt value function, Q_{π_θ}, can be seen in Equation 5.

- **Advantage Actor-Critic (A2C)** [10]:

  This is an on-policy actor-critic algorithm that makes use of the advantage function. An expression for the gradient of A2C's objective function (which uses the advantage function) can be seen in Equation 10.

- **Proximal Policy Optimisation (PPO)** [37]:

  This is an on-policy actor-critic algorithm which maximises the objective function below [37]:

$$J^{CLIP} = \mathbb{E}_{\pi_\theta(a_t|s_t)} \left[\min \left(b_t(\theta) A_t, \text{clip} \left(b_t(\theta), 1-\epsilon, 1+\epsilon \right) A_t \right) \right] \tag{35}$$

  Here, A_t is the advantage function (seen in Equation 10). $b_t(\theta)$ is the probability ratio $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ with θ_{old} referring to the previous policy parameters and ϵ is a hyperparameter with $\epsilon < 1$.[104] The use of this objective function regularises the updates of policy parameters θ by constraining the magnitude

[101] As seen in section 3.2.
[102] All these algorithms are model-free, online, single-agent RL algorithms.
[103] See section 2.1.1.2
[104] Schulman et al. [37] suggest $\epsilon = 0.2$.

of update steps.[105] This is constrained by placing upper and lower bounds on the magnitude of the gradient update steps so that large gradient steps cannot be taken. The clipping function having $clip(b_t, 1-\epsilon, 1+\epsilon) = \min(\max(b_t, 1-\epsilon), 1+\epsilon)$ achieves this by placing lower bound $1-\epsilon$ and upper bound $1+\epsilon$ on the probability ratio b_t.

These algorithms serve as our alternatives in the MCDA process such that $\mathcal{L} = \{\text{DQN, A2C, PPO}\}$. We choose the above three algorithms in part because we make use of RL Baselines Zoo, by Raffin [202], and these three are amongst the choices of implemented algorithms on a variety of tasks in this library. These three algorithms also offer a range of performances from the state of the art (PPO) to the more canonical (A2C and DQN) which is useful when trying to elicit a highly variable dataset for testing an evaluation procedure. Furthermore, these algorithms are offered by Raffin [202] on a variety of more light-weight discrete MDP RL tasks, which we seek due to limited computational resources.

4.1.2 Task Selection

Next we choose three tasks for algorithm evaluation, all of which can be found in the *Gymnasium* library [203]. We choose the tasks as they are relatively computationally light-weight due to them being considered easier than many others available in Gymnasium. The tasks are also easily implemented with RL Baselines Zoo [202] and provide variable performances for our three algorithms which we desire to effectively demonstrate our evaluation method.[106] Two of the tasks, Cart Pole and Acrobot, are considered to be *classic control* tasks by Gymnasium and are based on 'real-world' problems [203]. The third, Lunar Lander, is part of the collection of toy games in the Box 2D set of tasks in Gymnasium. These three tasks serve as our criteria in our MCDA process: $\Theta = \{\text{Cart Pole, Acrobat, Lunar Lander}\}$. The tasks are described as follows:

- **Cart Pole** [203]:

  This task originally appears in Barto et al. [205]. It involves a cart trying to balance a pole in the upright position. The cart is able to move left and right to achieve this, meaning that its action space is a one dimensional discrete action space corresponding to pushing the cart left or right (with a fixed force). The state space is four dimensional and continuous. It includes the cart's position and velocity as well as the pole's angular position and angular velocity. The reward function assigns a reward of $+1$ for every timestep that the task does not terminate. Termination occurs when the pole angle is greater than $12°$ off of the vertical, when the cart position is greater than 2.4 units off of the central position, or after 500 timesteps (corresponding to a maximum reward of 500).

- **Acrobot** [203]:

  This task is based on work in Sutton [206] and Sutton and Barto [7]. The task consists of a double jointed pendulum hanging downward with its top joint fixed. The agent must apply torques to the pendulum to swing the free end above a given height. The task has a one dimensional discrete action space corresponding to the torque that the agent can apply to the pendulum. It can apply a torque of -1, $+1$ or 0. The task has a six dimensional continuous state space. This corresponds to the angles and angular velocities of the pendulum's joints. The reward function assigns a reward of -1 for each timestep until a threshold of -500 is reached. Reaching the target height of $-\cos(\theta_1) - \cos(\theta_1 + \theta_2) > 1$ assigns a reward of 0 and terminates the episode.[107] This task poses a challenge for exploration as random actions are unlikely to succeed [204].

[105] This is intended to avoid phenomena known as catastrophic forgetting, whereby large changes to θ (in parameter space) may decay performance significantly.

[106] The combination of these three tasks with two of our aforementioned three chosen algorithms (DQN and A2C) can be seen in Millidge [204]. This paper demonstrates the variable performance of these two algorithms on these tasks. For example A2C can be seen to outperform DQN on Cart Pole, whereas DQN outperforms A2C on Lunar Lander.

[107] Here θ_1 is the angle of the first joint and θ_2 is the relative angle of the second joint with respect to the first [203]. $\theta_2 = 0$ implies that the joints are oriented at the same angle as one another.

- **Lunar Lander** [203]:

  In this task the algorithm must learn to land a spacecraft on a designated landing pad which is identified by two flags. The spacecraft is equipped with an infinite fuel supply and landing outside the designated area is possible. There is a one dimensional, discrete action space where the agent can choose to do nothing, fire its main engine, fire a leftward orientation engine or fire a rightward orientation engine. There is an eight dimensional state space which supplies the coordinates of the spacecraft, its linear velocities, angular velocities and two booleans variables that indicate whether each of its two legs are in contact with the ground or not. Rewards of +10 are assigned for each standing leg, +100 for landing safely and -100 for crashing. Rewards are incremented depending on how far the spacecraft is from the landing pad and depending on how fast the spacecraft is moving. Rewards are also decreased if the spacecraft's angle of orientation with respect to the ground is tilted. Rewards are decreased by 0.03 for each timestep that a side engine is firing and by 0.3 for each timestep that the main engine is firing. An episode is considered successful if the algorithm achieves a return of +200 or more. Episodes are terminated if the spacecraft crashes or moves greater than 1 unit out of view.

4.1.3 Data Collection

The data collection subsection describes how to collect data from running our set of algorithms $\mathcal{L}$ on our set of tasks Θ for the purpose of algorithm comparison via an empirical performance evaluation. Our data collection process for this proof-of-concept test can be seen in our supplementary data collection notebook.[108] Furthermore the collected data can be found in our supplementary repository.[109]

4.1.3.1 Hyperparameters

For the purpose of this proof-of-concept test of our evaluation procedure, we use the default hyperparameters provided by the implementations of the RL Baselines Zoo library [202]. We report these for ease of reproducibility. We use the reporting template, inspired by the supplementary material of Gorsane et al. [1] for ease of reporting. The parameters used can be found in Tables 5, 6 and 7, which correspond to each of the three tasks, Cart Pole, Acrobot and Lunar Lander respectively. The chosen hyperparameters are tuned by Raffin [202].[110]

4.1.3.2 Runs per Task, Duration of Runs and Evaluation Frequency Across Runs

To collect the performance data of our algorithms on the three tasks we choose to do five runs for each algorithm on each task. This gives $N_j = 5 \ \forall j \in \Theta$. We run each task for $T = 100000$ timesteps. We evaluate each algorithm on each task and each run every $t^* = 10000$ timesteps over $E = 10$ evaluation episodes to establish robust performance metrics for the check-pointed policies at those timesteps. At the end of the T timesteps on each run $n \in \{1, 2, 3, 4, 5\}$ on each task $j \in \Theta$ we calculate the final metric $x_{n,j}(k)$ as the average of $E = 10$ evaluation episodes of the check-pointed policy at that time. This acts as our performance score data for algorithms $k \in \mathcal{L}$. All these evaluation details appear in Tables 5, 6 and 7.

The sample efficiency curves displaying the training runs of the three algorithms on the three tasks can be seen in the results section, section 4.2.[111]

[108] https://github.com/marlEvalDissertation/marlEvalDiss/blob/main/Guideline_Data_Collection.ipynb
[109] https://github.com/marlEvalDissertation/marlEvalDiss/tree/main/RL_data
[110] For DQN, Raffin [202] uses the defaults obtained from Mnih et al. [35] except for the optimiser and learning rate.
[111] See Figure 21 in section 4.2 as well as Figures 24, 25 and 26 in the appendix.

Table 5: Experimental details for our algorithm comparison of the set of algorithms $\mathcal{L}$ on **Cart Pole** Version 1 [203]. Hyperparameters are obtained as defaults from Raffin [202] which uses Raffin et al. [207].

Experimental setup	**DQN**	**A2C**	**PPO**
Hyperparameters			
Discount factor gamma	0.99	0.99	0.98
Batch size	64	NA	256
Number of Epochs	NA	NA	20
Buffer size	100000	NA	NA
Minimum replay buffer size before updating	100	NA	NA
N steps bootstrapping	NA	5	32
Target network update period	250	NA	NA
Gradient steps after rollouts	128	NA	NA
Training Frequency	256	NA	NA
ϵ schedule (Decay steps, ϵ start, ϵ min)	(0.16,1,0.04)	NA	NA
Network architecture	MLP Policy	MLP Policy	MLP Policy
Network Layer size	(256,256)	None	None
Normalise Advantage	NA	False	False
Seed range	$[0, 2^{32} - 1]$	$[0, 2^{32} - 1]$	$[0, 2^{32} - 1]$
Tau (soft updates)	1	NA	NA
Buffer class	None	None	None
Buffer Args	None	None	None
Optimise Buffer Memory	False	NA	NA
Number of parallel Envs	NA	8	8
Learning rate	0.0023	0.0007	0.001
Max Value for Gradient Clipping	10	0.5	10
Clip range	NA	NA	0.2
Value function clipping range	NA	NA	None
$TD(\lambda)$ value	NA	1	0.8
Build network at creation	True	True	True
Optimiser	Adam	RMSProp	Adam
RMSProp Epsilon	NA	0.00005	NA
Value function Coefficient	NA	0.5	0.5
State Dependent Exploration	NA	False	False
Target KL Divergence	NA	Na	None
Computational resources	Google Colab Notebook		
Average Wall-clock time per algorithm	202.3s	68.8s	100.8s
CPUs per experiment	2	2	2
GPU per experiment	1	1	1
RAM per experiment	15GB	15GB	15GB
Device	Auto	Auto	Auto
Evaluation protocol			
Algorithm Training Runs N	5	5	5
Total training (timesteps) T	100000	100000	100000
Evaluation interval (timesteps) t^*	10000	10000	10000
Independent evaluation episodes E	10	10	10
Run Evaluation Metric $x_{n,j}(k)$	Final Metric	Final Metric	Final Metric
Local aggregation method	Mean	Mean	Mean
Global aggregation method	MCDA	MCDA	MCDA
Exploration behaviour (coef)	Epsilon-Greedy	Entropy(0)	Entropy (0)

Table 6: Experimental details for our algorithm comparison of the set of algorithms $\mathcal{L}$ on **Acrobot** Version 1 [203]. Hyperparameters are obtained as defaults from Raffin [202] which uses Raffin et al. [207].

Experimental setup	**DQN**	**A2C**	**PPO**
Hyperparameters			
Discount factor gamma	0.99	0.99	0.99
Batch size	128	NA	256
Number of Epochs	NA	NA	4
Buffer size	50000	NA	NA
Minimum replay buffer size before updating	1000	NA	NA
N steps bootstrapping	NA	5	256
Target network update period	250	NA	NA
Gradient steps after rollouts	-1	NA	NA
Training Frequency	256	NA	NA
ϵ schedule (Decay steps, ϵ start, ϵ min)	(0.12,1,0.1)	NA	NA
Network architecture	MLP Policy	MLP Policy	MLP Policy
Network Layer size	(256,256)	None	None
Normalise Advantage	NA	True	True
Seed range	$[0, 2^{32} - 1]$	$[0, 2^{32} - 1]$	$[0, 2^{32} - 1]$
Tau (soft updates)	1	NA	NA
Buffer class	None	None	None
Buffer Args	None	None	None
Optimise Buffer Memory	False	NA	NA
Number of parallel Envs	NA	16	16
Learning rate	0.00063	0.0007	0.0003
Max Value for Gradient Clipping	10	0.5	10
Clip range	NA	NA	0.2
Value function clipping range	NA	NA	None
$TD(\lambda)$ value	NA	1	0.94
Build network at creation	True	True	True
Optimiser	Adam	RMSProp	Adam
RMSProp Epsilon	NA	0.00005	NA
Value function Coefficient	NA	0.5	0.5
State Dependent Exploration	NA	False	False
Target KL Divergence	NA	Na	None
Computational resources	Google Colab Notebook		
Average Wall-clock time per algorithm	585.3s	43.3s	61.6s
CPUs per experiment	2	2	2
GPU per experiment	1	1	1
RAM per experiment	15GB	15GB	15GB
Device	Auto	Auto	Auto
Evaluation protocol			
Algorithm Training Runs N	5	5	5
Total training (timesteps) T	100000	100000	100000
Evaluation interval (timesteps) t^*	10000	10000	10000
Independent evaluation episodes E	10	10	10
Run Evaluation Metric $x_{n,j}(k)$	Final Metric	Final Metric	Final Metric
Local aggregation method	Mean	Mean	Mean
Global aggregation method	MCDA	MCDA	MCDA
Exploration behaviour (coef)	Epsilon-Greedy	Entropy(0)	Entropy (0)

Table 7: Experimental details for our algorithm comparison of the set of algorithms $\mathcal{L}$ on **Lunar Lander** Version 2 [203]. Hyperparameters are obtained as defaults from Raffin [202] which uses Raffin et al. [207].

Experimental setup	**DQN**	**A2C**	**PPO**
Hyperparameters			
Discount factor gamma	0.99	0.995	0.999
Batch size	128	NA	64
Number of Epochs	NA	NA	4
Buffer size	50000	NA	NA
Minimum replay buffer size before updating	0	NA	NA
N steps bootstrapping	NA	5	1024
Target network update period	10	NA	NA
Gradient steps after rollouts	-1	NA	NA
Training Frequency	4	NA	NA
ϵ schedule (Decay steps, ϵ start, ϵ min)	(0.12,1,0.1)	NA	NA
Network architecture	MLP Policy	MLP Policy	MLP Policy
Network Layer size	(256,256)	None	None
Normalise Advantage	NA	False	False
Seed range	$[0, 2^{32} - 1]$	$[0, 2^{32} - 1]$	$[0, 2^{32} - 1]$
Tau (soft updates)	1	NA	NA
Buffer class	None	None	None
Buffer Args	None	None	None
Optimise Buffer Memory	False	NA	NA
Number of parallel Envs	NA	8	16
Learning rate	0.00063	0.00083	0.0003
Max Value for Gradient Clipping	10	0.5	10
Clip range	NA	NA	0.2
Value function clipping range	NA	NA	None
$TD(\lambda)$ value	NA	1	0.98
Build network at creation	True	True	True
Optimiser	Adam	RMSProp	Adam
RMSProp Epsilon	NA	0.00005	NA
Value function Coefficient	NA	0.5	0.5
State Dependent Exploration	NA	False	False
Target KL Divergence	NA	Na	None
Computational resources	Google Colab Notebook		
Average Wall-clock time per algorithm	565.1s	91.6s	86.4s
CPUs per experiment	2	2	2
GPU per experiment	1	1	1
RAM per experiment	15GB	15GB	15GB
Device	Auto	Auto	Auto
Evaluation protocol			
Algorithm Training Runs N	5	5	5
Total training (timesteps) T	100000	100000	100000
Evaluation interval (timesteps) t^*	10000	10000	10000
Independent evaluation episodes E	10	10	10
Run Evaluation Metric $x_{n,j}(k)$	Final Metric	Final Metric	Final Metric
Local aggregation method	Mean	Mean	Mean
Global aggregation method	MCDA	MCDA	MCDA
Exploration behaviour (coef)	Epsilon-Greedy	Entropy(0.00001)	Entropy (0.01)

4.1.4 Preference Modelling

4.1.4.1 Constructing Partial Value Functions

We now need to construct partial value functions $g_j : x_{n,j}(k) \longrightarrow [0, 100]$ that take performance scores $x_{n,j}(k)$ for algorithms $k \in \mathcal{L}$ and runs $n \in \{1, ..., N_j\}$ on each task $j \in \Theta$ and map these to $[0, 100]$. These must be constructed for each of our three RL tasks so as to account for the non-linear scaling of task difficulty with performance scores [21, 195].

In section 3.1 we recommended that partial value functions be constructed by either the creators of the RL tasks or domain experts due to the necessity to be particularly familiar with algorithm behaviour on these tasks, as well as the need to be familiar with how the algorithm behaviours interact with the tasks' assigned reward functions. This would minimise the risk of judgemental errors affecting the construction of such functions. In this dissertation we are constrained to our own limited understanding of our RL tasks, their reward functions and agent behaviour on these. Hence, we attempt to construct partial value functions only as an example of their use and with regard for our shortcomings. Despite this we note the insights of Stewart [197, 198] who suggests that the construction of such value functions is robust to the judgemental errors of the decision-maker provided that points are within approximately 10% of their 'true' values and consistency checks are done.

The first step in constructing our partial value functions is to note that we will have monotonically increasing functions. This is since we have preference (in RL) for greater amounts of reward. We then identify two reference points per RL task. For these we find the minima and maxima of the reward functions of each task. These we use as our *global minima* and *global maxima* [195].

We choose our partial value functions to be constrained to $[0, 100]$. We thus assign the minima values of 0 and the maxima values of 100 such that $g_j(x_{min}) = 0$ and $g_j(x_{max}) = 100$, $\forall j \in \Theta, k \in \mathcal{L}$ and $n \in \{1, ..., N_j\}$. The following global minima and maxima are found for our algorithm runs:[112]

- **Cart Pole**: $g_{cartpole}(0) = 0$ and $g_{cartpole}(500) = 100$
- **Acrobot**: $g_{acrobot}(-500) = 0$ and $g_{acrobot}(-50) = 100$
- **Lunar Lander**: $g_{lunarlander}(-3000) = 0$ and $g_{lunarlander}(300) = 100$

We now use the partial value function methods by von Winterfeldt and Edwards [192] for constructing the partial value functions.[113] In order to imitate an expert's knowledge of algorithm performance on these three tasks we consider our algorithm sample efficiency curves (seen in Figure 21 in section 4.2, and in Figures 24, 25 and 26 in the appendix) as well as the sample efficiency curves of various algorithms on the same three tasks as seen in Millidge [204] (see Figures 15, 17 and 19). Our consideration of the algorithm performances on these plots acts to inform our preferences when using the value function construction methods. We note that our considerations drawn from these figures may be different than those of an expert. Moreover, an expert's opinions may result in entirely different value functions being constructed. However, we pursue this process merely as an example of such constructions. We also note that the MCDA approach is meant to capture the subjectivity of a decision-maker's preference for different levels of reward. Constructing such value functions in an iterative process, using experts, performing consistency checks, forming consensus opinions if multiple experts are present, and using the specific elicitation methods described are all useful practises to ensure robust construction.[114]

[112] Any extreme outliers that are either below these minima or above the maxima will be assigned values of 0 or 100 respectively.

[113] These methods are elaborated on in sections 2.4.4 and 3.1.3.1.

[114] Furthermore it may be interesting and useful to the engineering process if decision-makers construct different value

- **Cart Pole**:

  Following the difference method for constructing partial value functions by von Winterfeldt and Edwards [192] we start by defining an arbitrary unit level of value u (see sections 2.4.4 and 3.1.3.1). As a reminder, u represents the amount of preferential value that we allocate to an increase in performance of an algorithm that increases its performance over between one fifth and one tenth of the domain of our partial value function (the range of the reward function). This interval we define in relation to one of our reference points. As we suspect a convex value function [197] we define this in relation to our reference point $(x_{max}, 100)$.[115] We define u over the interval $[450, 500]$ such that we have selected $\kappa \in [\frac{1}{10}, \frac{1}{5}]$ as desired, and such that u is defined by the increase in value over this interval. This yields:

$$\begin{aligned} u &= g_{cartpole}\left(x_{\max}\right) - g_{cartpole}\left(a_{\zeta-1,\max}\right) \\ &= g_{cartpole}\left(500\right) - g_{cartpole}\left(450\right) \\ &\implies a_{\zeta-1,\max} = 450 \\ &\implies [a_{\zeta-1,\max}, x_{\max}] = [450, 500] \text{ for finite } \zeta \in \mathbb{N} \end{aligned} \tag{36}$$

  The method by von Winterfeldt and Edwards [192] then asks of the domain expert to find the adjacent interval such that an increase over the range of that interval will have a value equal to u. Studying Figure 15 and Figure 21 **(a)** informs our choices and thereby acts as proxy for domain experience in the absence of an 'expert'. From these figures it would appear that an increase over the range $[450, 500]$ is equivalent to an increase over the range $[350, 450]$ in terms of the level of difficulty of achievement for an algorithm. We arrive at this choice by noting that two algorithms achieve a score of at least 350 after training, whereas only one achieves a score of approximately 450 in Figure 15. Only one algorithm achieves a score of at least 500 in Figure 21 **(a)** whereas two achieve a score of at least 450. Looking at the training time in Figure 15 the three algorithms shown stabilise around scores of 450, 400 and 250 respectively. One could say from this that an increase from 400 to 450 is more attainable and less difficult to achieve than an increase from 450 to 500 which appears more difficult as none of the algorithms achieve this with stability. It appears approximately equally difficult for the active inference algorithm (green) to increase to a level of 450 from a score of approximately 350 as for it to increase to 500 from 450. Hence, it would appear from these plots that a difficulty of increasing over the interval $[450, 500]$ is approximately equivalent to the difficulty of increasing over the interval $[350, 450]$. This gives our third reference point for the partial value function $g_{cartpole}$ as $(350, g_{cartpole}(350))$.

  Continuing similarly and studying these plots further, we conclude that an increases over the intervals $[x_{\min}, 250]$ and $[250, 350]$ are approximately equal to u.[116]

  Thus we have chosen the interval endpoints that make up intervals A_α such that:

$$\begin{aligned} &\bigcup_{\alpha \in I} A_\alpha = [x_{\min}, x_{\max}] \text{ for } I = \{1, ..., 4\} \\ &\bigcup_{\alpha \in I} A_\alpha = [x_{\min}, 250] \cup [250, 350] \cup [350, 450] \cup [450, x_{\max}] \end{aligned} \tag{37}$$

functions for the same RL task. This process therefore offers the possibility for extra information emerging from the evaluation process.

[115] This is as opposed to $(x_{min}, 0)$ since we desire a small interval (of between a fifth and a tenth of the domain) to define u over. A convex value function will mean that choosing such a small interval from the reference point $(x_{min}, 0)$ will result in a partial value function that is complex to construct as it will have many reference points. As we aim to construct a partial value function with as few as five reference points [195], we construct this using an interval defined from $(x_{max}, 100)$.

[116] Performing consistency checks with one or more 'experts' by asking similar questions about levels of performance leads to the improved robustness of the constructed value functions [194]. It is therefore essential to perform such consistency checks when constructing such partial value functions. One could use another method of preference elicitation such as the bisection method (from section 2.4.4) for the purpose of performing consistency checks. One could also compare partial value functions constructed by several experts and aggregate across them or choose one by consensus. While the consistency checks are important they are, however, not done explicitly in this dissertation where we merely provide an example of such a construction to illustrate the procedure.

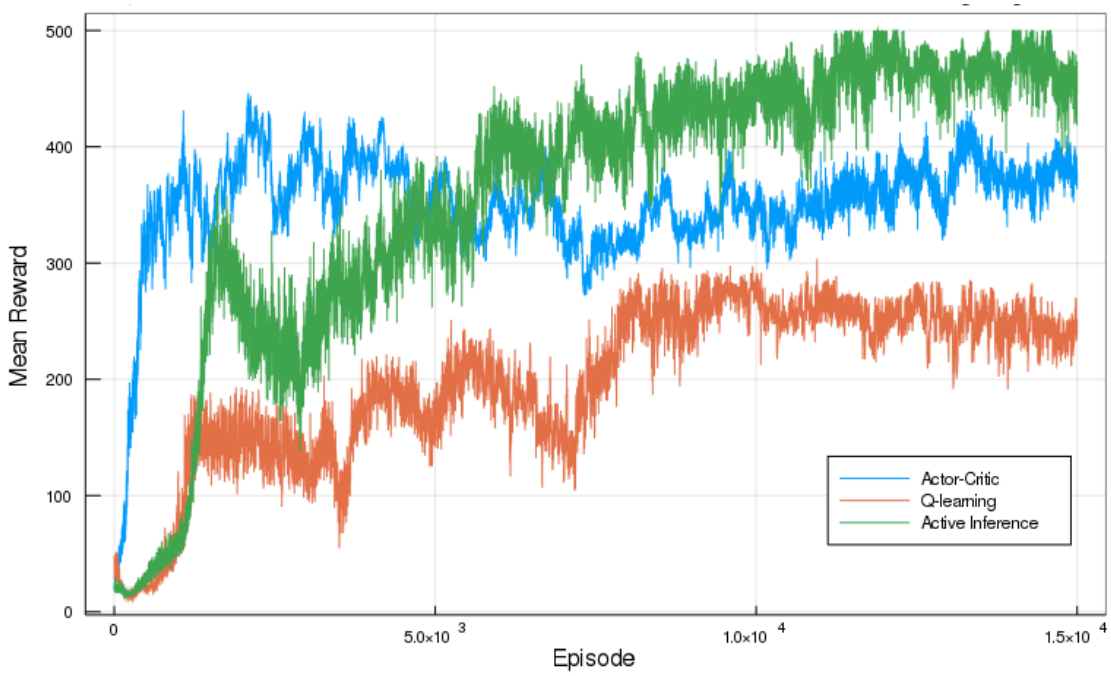

Figure 15: This plot appears in Millidge [204] and depicts the performance of three deep RL algorithms on **Cart Pole**. The three algorithms include a deep active inference algorithm by Millidge [204] (green), A2C (blue) and DQN (orange).

This gives $g_{cartpole}(0) = 0$, $g_{cartpole}(250) = 25$, $g_{cartpole}(350) = 50$, $g_{cartpole}(450) = 75$ and $g_{cartpole}(500) = 100$. Using these points with the piecewise partial value function equation (see Equation 29) gives the resulting partial value function (plotted in Figure 16):[117]

$$g_{cartpole}(x_{n,\text{cartpole}}(k)) = \begin{cases} 0 & \text{if } x \in (-\infty, 0) \\ \frac{25}{250}(x) & \text{if } x \in [0, 250) \\ 25 + \frac{25}{350-250}(x - 250) & \text{if } x \in [250, 350) \\ 50 + \frac{25}{450-350}(x - 350) & \text{if } x \in [350, 450) \\ 75 + \frac{25}{500-450}(x - 450) & \text{if } x \in [450, 500] \\ 100 & \text{if } x \in (500, \infty) \end{cases} \tag{38}$$

- **Acrobot**:

  For Acrobot we demonstrate a second method discussed in the literature for eliciting information from domain experts to construct partial value functions. We make use of the bisection method by von Winterfeldt and Edwards [192].[118] This asks of us to consider the performance score range between x_{max} and x_{min} and to identify the point $x = x_{50}$ where an increase in performance from x_{min} to x_{50} would be equal in preferential value to an increase from x_{50} to x_{max} i.e.

  $$g_{acrobot}(x_{max}) - g_{acrobot}(x_{50}) = g_{acrobot}(x_{50}) - g_{acrobot}(x_{min})$$

  Thereafter we identify points x_{25} and x_{75} which have

  $$g_{acrobot}(x_{\max}) - g_{acrobot}(x_{75}) = g_{acrobot}(x_{75}) - g_{acrobot}(x_{50})$$

  and

  $$g_{acrobot}(x_{50}) - g_{acrobot}(x_{25}) = g_{acrobot}(x_{25}) - g_{acrobot}(x_{min})$$

[117] See our data aggregation notebook https://github.com/marlEvalDissertation/marlEvalDiss/blob/main/Guideline_Data_Aggregation.ipynb

[118] See sections 2.4.4 and 3.1.3.1.

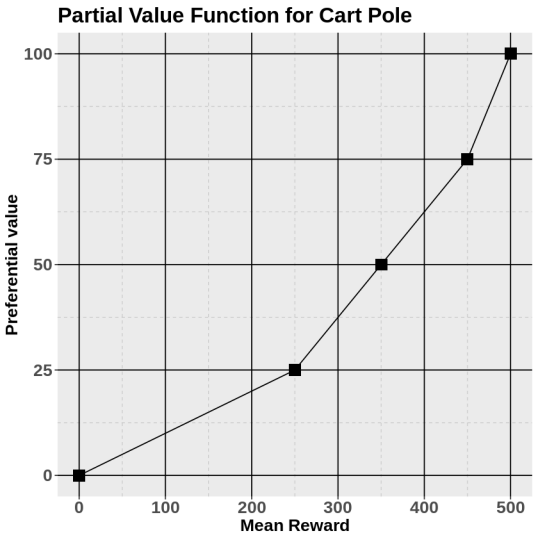

Figure 16: This figure depicts the construction of a partial value function measuring the strength of preference of a user for the different amounts of reward achieved by a deep RL algorithm on the **Cart Pole** task. This is constructed using the difference method by von Winterfeldt and Edwards [192]. The function expresses preferential value on the scale [0, 100].

We study Figures 17 and 21 **(b)** to inform these choices. This yields a choice of $x_{50} = -150$. To arrive at this choice, we note that for the Acrobot task (as seen in Figure 21 **(b)**) algorithm score increases from scores of -500 to around -150 are quickly obtained with very few training steps, indicating a level of easiness for the algorithms. In contrast, increasing beyond -150 takes comparatively many more timesteps. In Figure 17, performance increases are quickly obtained to levels of around -150 for two of the three algorithms (red and blue). However, the performance score of the policy gradient algorithm (red) decays from -150, and the performance of DQN (green) takes relatively long to reach a score of around -200. From this we select x_{50} to be around -150 and construct our partial value function such that an increase from -500 to -150 is equivalent to an increase from -150 to -50, in terms of preferential value.

We continue similarly to choose $x_{25} = -200$ and $x_{75} = -75$. In selecting x_{75}, we note that the algorithms are able to improve more easily from -150 to -100 than from -100 to -50. This alludes to a greater level of difficulty when moving from scores of -100 to -50 than from -150 to -50. This means we must assign more value to the increase from -100 to -50. As no algorithm is able to achieve scores of -50 in either figures studied, we assign more than twice the value to the increase from -100 to -50 and say that $x_{75} = -75$.

In selecting x_{25} we note that in Figure 17 the active inference algorithm (blue) achieves a score of around -250 in approximately half the timesteps as it needs to achieve -150. DQN (green) achieves a score of around -200 at the end of training. Halfway through its training it achieves a score of around -250. This indicates that moving from -500 to -250 may be considered easier than moving from -250 to -150 and hence $x_{25} > -250$. We hence choose $x_{25} = 200$ for our example and note, once again, the importance of consistency checks when constructing such a value function for true evaluations.

Our resulting partial value function thus has that: $g_{acrobot}(-500) = 0$, $g_{acrobot}(-200) = 25$, $g_{acrobot}(-150) = 50$, $g_{acrobot}(-75) = 75$ and $g_{acrobot}(-50) = 100$. This results in the following piece-wise linear partial value function (plotted in Figure 18):[119]

[119] See our data aggregation notebook https://github.com/marlEvalDissertation/marlEvalDiss/blob/main/Guideline_Data_Aggregation.ipynb

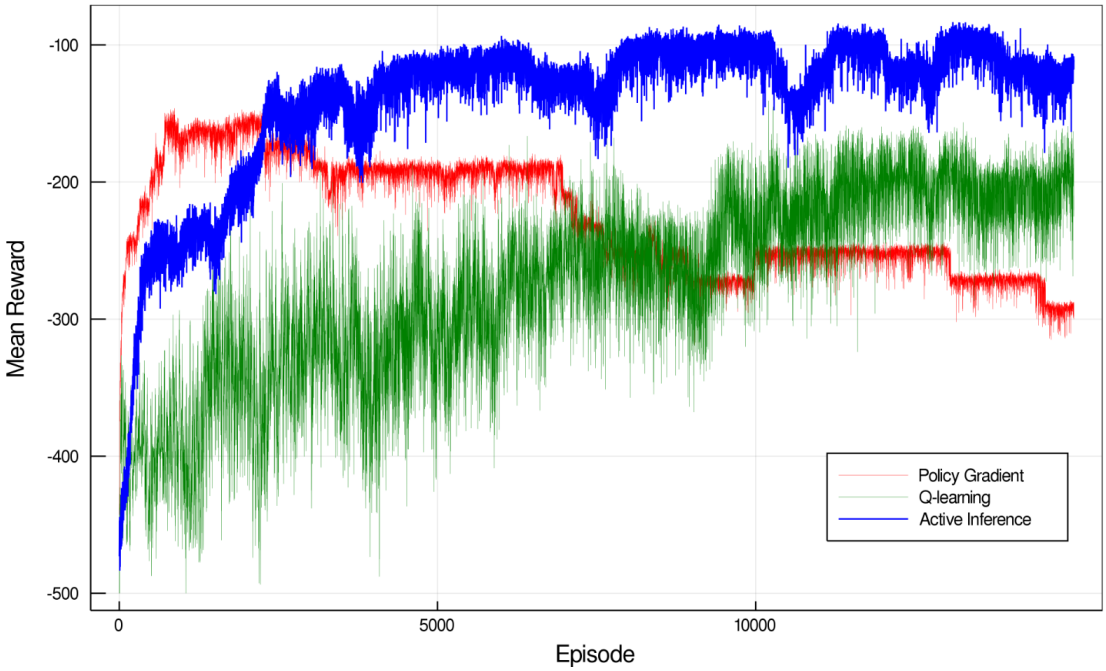

Figure 17: This plot appears in Millidge [204] and depicts the performance of three deep RL algorithms on **Acrobot**. The three algorithms include a deep active inference algorithm by Millidge [204] (blue), DQN (green) and a vanilla policy gradient algorithm (red).

$$g_{acrobot}(x_{n,\text{acrobot}}(k)) = \begin{cases} 0 & \text{if } x \in (-\infty, -500) \\ \frac{25}{-200+500}(x+500) & \text{if } x \in [-500, -200) \\ 25 + \frac{25}{-150+200}(x+200) & \text{if } x \in [-200, -150) \\ 50 + \frac{25}{-75+150}(x+150) & \text{if } x \in [-150, -75) \\ 75 + \frac{25}{-50+75}(x+75) & \text{if } x \in [-75, -50] \\ 100 & \text{if } x \in (-50, \infty) \end{cases} \tag{39}$$

- **Lunar Lander**:

  Similarly to Acrobot, we use the bisection method by von Winterfeldt and Edwards [192] for constructing our partial value function for Lunar Lander. We select x_{25}, x_{50} and x_{75} by considering Figure 19 from Millidge [204] and Figure 21 **(c)** which displays the results of our algorithm runs. These are again used for informative purposes to construct our example value functions in order to replace the use of 'expert' knowledge which is necessary for constructing such value functions in true evaluations. From this we select $\{x_{min}, x_{25}, x_{50}, x_{75}, x_{max}\} = \{-3000, -200, 0, 200, 300\}$.

  We arrive at these choices by noting the following: Lunar Lander's optional performance, according to Towers et al. [203], is anything above 200. We observe that only one algorithm in either Figure 19 or Figure 21 **(c)** achieves this. Four of the algorithms across the two plots have final scores of around -200, and two of them have scores around 0. Thus, we select a bisection point of $x_{50} = 0$ which suggests that we are indifferent between increases in performance from -3000 to 0 as for increases from 0 to 300. As only one algorithm is able to achieve a score of above around 0, by achieving a score of around 200, we view the increase from 0 to 200 as particularly difficult to achieve similarly to the difficulty of improving beyond 200 to 300. Since we view anything beyond a score of 200 to be optimal we hold a similar preference for the increase from 0 to 200 as for 200 to 300. Hence, we choose $x_{75} = 200$. Furthermore most algorithms very quickly achieve scores greater than -500. This implies an easy level of difficulty to move to this score level when starting from -3000. Hence, we need $x_{25} > -500$. As four of the 6 algorithms tend to performance scores of -200 after training, and

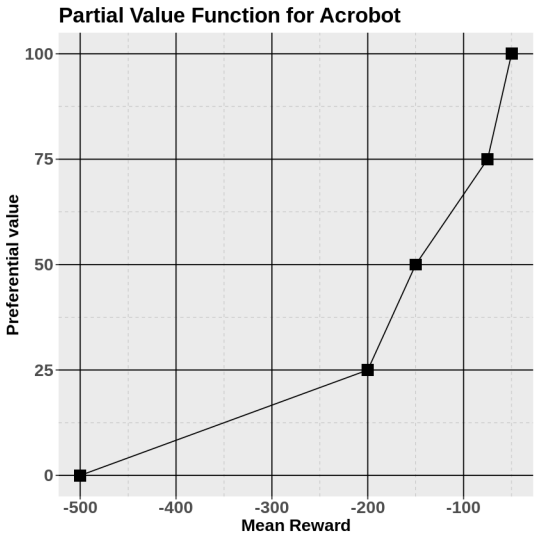

Figure 18: This figure depicts the construction of a partial value function measuring the strength of preference of a user for the different amounts of reward achieved by a deep RL algorithm on the **Acrobot** task. This is constructed using the bisection method by von Winterfeldt and Edwards [192]. The function expresses preferential value on the scale [0, 100].

as this indicates a relatively high level of difficulty to move from -200 to 0, we view an increase from -3000 to -200 as equivalent to an increase from -200 to 0. From this we select $x_{25} = -200$.

This yields the following piece-wise linear partial value function (plotted in Figure 20):[120]

$$g_{lunarlander}(x_{n,\text{acrobot}}(k)) = \begin{cases} 0 & \text{if } x \in (-\infty, -3000) \\ \frac{25}{-200+3000}(x+3000) & \text{if } x \in [-3000, -200) \\ 25 + \frac{25}{200}(x+200) & \text{if } x \in [-200, 0) \\ 50 + \frac{25}{200}(x) & \text{if } x \in [0, 200) \\ 75 + \frac{25}{300-200}(x-200) & \text{if } x \in [200, 300] \\ 100 & \text{if } x \in (300, \infty) \end{cases} \tag{40}$$

4.1.5 Data Aggregation

Using the above partial value functions we turn to the additive value function model expressed in Equation 34. This takes algorithm performance scores of any one algorithm $k \in \mathcal{L}$, and aggregates these across runs and tasks. In order to perform such an aggregation, we must assign swing weights to each of our tasks.

4.1.5.1 Choosing Weights

We now make use of the swing weight method to select the relative weights q_j for all tasks $j \in \Theta$. Recall that the relative weights and the swing weight method are used in order to account for *both* the relative importance of each task with respect to each other and the extent to which the reward scales used discriminate between tasks. In this dissertation, we provide an example of one possible use of the swing weight method, again noting our lack of an expert view on the three RL tasks used.

[120] See our data aggregation notebook https://github.com/marlEvalDissertation/marlEvalDiss/blob/main/Guideline_Data_Aggregation.ipynb

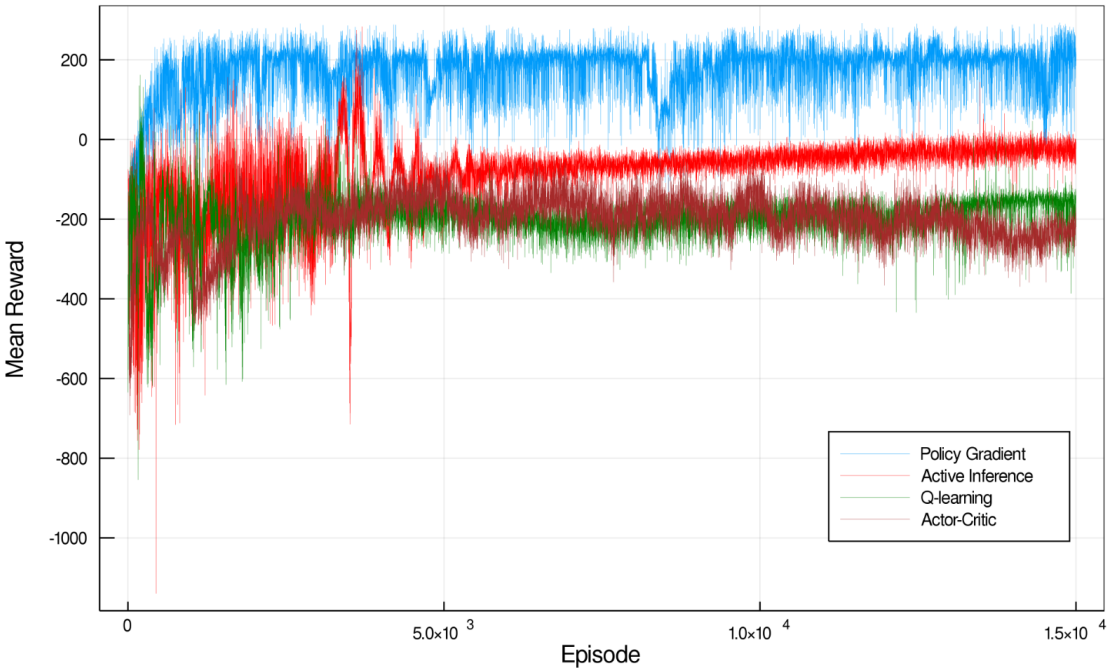

Figure 19: This plot appears in Millidge [204] and depicts the performance of four deep RL algorithms on **Lunar Lander**. The four algorithms include a deep active inference algorithm by Millidge [204] (red), a vanilla policy gradient algorithm (light blue), DQN (green) and A2C (maroon).

We begin by ranking the *swings* from worst (x_{min}) to best (x_{max}) performance on each task in $\Theta = \{\text{Cart Pole, Acrobat, Lunar Lander}\}$. We use Figure 21 as a reference point as to the performances of our three algorithms on the tasks. We note from this that all three algorithms achieve near optimal performance on Acrobot in comparatively few timesteps. Only one algorithm is able to converge and achieve optimal performance in the Cart Pole task, and none in Lunar Lander. This suggests that our algorithms find Acrobot comparatively easy to achieve high performance scores on and Lunar Lander comparatively more challenging. From this we rank the magnitudes of the weights: $q_{lunarlander} > q_{cartpole} > q_{acrobot}$.

The swing weight method then suggests we assign values to weights by assessing the relative importance of swings in performance of all tasks with respect to the largest weight. We assign a value of 100 to the largest weight such that $q_{lunarlander} = 100$. We then perform $M - 1 = |\Theta| - 1 = 2$ comparisons against Lunar Lander's weight to assign values to the relative importance that we place on the swings from worst to best performance. We do this by asking the decision-maker the question: *What is the relative value of a swing from worst to best for the other RL tasks as compared to a value of 100 for a swing from worst to best on Lunar Lander?* From this we assign $q_{cartpole} = 75$ and $q_{acrobot} = 50$.

For these assessments we again consider Figure 21 but this time we have the advantage of considering it in relation to our preferred levels of values as expressed by our partial value functions in Figures 16, 18 and 20. If a relevant swing for Lunar Lander is assigned a value of $q_{lunarlander} = 100$ we first consider the ratio $\frac{q_{cartpole}}{q_{lunarlander}} = \frac{q_{cartpole}}{100} < 1$. We think about an algorithm swinging its performance from worst to best on Lunar Lander in relation to the swing from worst to best on Cart Pole. If intuition told us that this ratio had that $\frac{q_{cartpole}}{q_{lunarlander}} = 0.75 \iff q_{cartpole} = 75$, then we can ask what this would mean in terms of performances of algorithms on the tasks. We can consider this in terms of two *indifference points* (in terms of preference) and therefore in terms of Equation 28 and the trade-off property discussed in section 2.4.5. Thinking about hypothetical indifference points and their associated levels of performance, in terms of our levels of preference for them, assists us in understanding what our choices of weights mean. One can similarly consider this for Acrobot.

Given these weights we must then perform consistency checks against a reference task as recommended by Stewart [195]. If we choose our reference criterion to be Acrobot, then we ask of the decision-maker

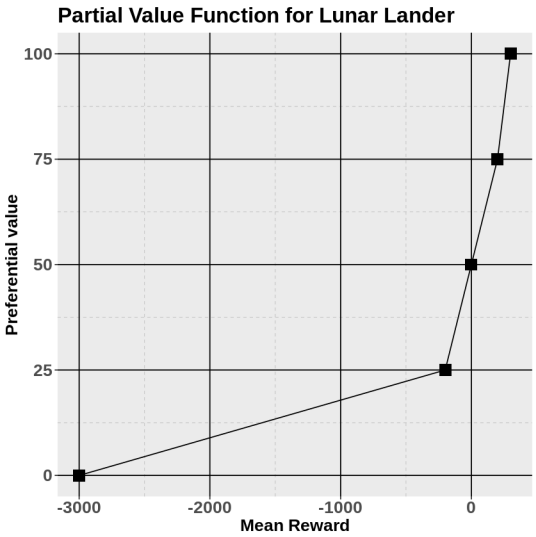

Figure 20: This figure depicts the construction of a partial value function measuring the strength of preference of a user for the different amounts of reward achieved by a deep RL algorithm on the **Lunar Lander** task. This is constructed using the bisection method by von Winterfeldt and Edwards [192]. The function expresses preferential value on the scale [0, 100].

to ensure that their choices of weights make sense. Hence, we consider ratios of weights $\frac{q_j}{q_{acrobot}} > 1$ for $j \in \{\text{Cart Pole, Lunar Lander}\}$. This yields ratios of $\frac{q_{lunarlander}}{q_{acrobot}} = 2$ and $\frac{q_{cartpole}}{q_{acrobot}} = 1.5$. One can then revert back on their choice of weights if the consistency checks do not agree with their preferences. Further consistency checks can be done in our case since we have only three tasks. Hence, we choose Cart Pole as a reference task and consider the ratios $\frac{q_{lunarlander}}{q_{cartpole}} = 1.33 > 1$ and $\frac{q_{acrobot}}{q_{cartpole}} = 0.67 < 1$. We find these to be consistent with our preferences.

Finally, we normalise our relative weights q_j to sum to 1 such that:

$$(q_{lunarlander}, q_{cartpole}, q_{acrobot}) = (\frac{4}{9}, \frac{3}{9}, \frac{2}{9}) \approx (0.44, 0.33, 0.22) \tag{41}$$

4.1.5.2 Using Additive Value Functions

We combine our weights in Equation 41 with our partial value functions and the additive value function in Equation 34 to provide an aggregated empirical performance score for our algorithm performances, aggregated across tasks. Here, we note that since we have five runs per task for each task, we have $N_j = 5$, $\forall j \in \Theta$.[121]

If we denote our algorithm performance scores $x_{n,j}$ for each task $j \in \Theta$ as a vector such that:

$$(x_{j,1}(k), ..., x_{j,5}(k)) = \mathbf{x}_j(k) = \mathbf{x}_j \tag{42}$$

and if we denote our aggregate algorithm performance scores y as:

$$y(\mathbf{x}_{cartpole}, \mathbf{x}_{acrobot}, \mathbf{x}_{lunarlander}) = y(k) \tag{43}$$

[121] Note the discussion on weighting each of the runs per task in section 3.1.4.

Then for any algorithm k we have:

$$\begin{aligned} y(k) &= \frac{q_{\text{cartpole}}}{5}\sum_{n=1}^{5} g_{\text{cartpole}}\left(x_{\text{cartpole},n}\right) + \frac{q_{\text{acrobot}}}{5}\sum_{n=1}^{5} g_{\text{acrobot}}\left(x_{\text{acrobot},n}\right) + \frac{q_{\text{lunarlander}}}{5}\sum_{n=1}^{5} g_{\text{lunarlander}}\left(x_{\text{lunarlander},n}\right) \\ &= \frac{0.33}{5}\sum_{n=1}^{5} g_{\text{cartpole}}\left(x_{\text{cartpole},n}\right) + \frac{0.22}{5}\sum_{n=1}^{5} g_{\text{acrobot}}\left(x_{\text{acrobot},n}\right) + \frac{0.44}{5}\sum_{n=1}^{5} g_{\text{lunarlander}}\left(x_{\text{lunarlander},n}\right) \\ &= 0.0667\sum_{n=1}^{5} g_{\text{cartpole}}\left(x_{\text{cartpole},n}\right) + 0.0444\sum_{n=1}^{5} g_{\text{acrobot}}\left(x_{\text{acrobot},n}\right) + 0.0889\sum_{n=1}^{5} g_{\text{lunarlander}}\left(x_{\text{lunarlander},n}\right) \end{aligned} \tag{44}$$

We can now conduct our algorithm performance comparison using this function for aggregation.

4.2 Results: RL Algorithm Performances Using Our Guideline

Using the evaluation guideline provided in section 3.2 we have produced an example evaluation of three RL algorithms on three tasks. We have proceeded to make use of all aspects required by the guideline and now seek to present the results of this algorithm evaluation and compare the algorithms from this. In this section we present these results and begin to assess the final research question of this dissertation, on the effectiveness of the recommendations provided by the guideline.

Section 3.3 outlined how our guideline resolves the two issues (of normalisation and weighting) that we identified through the answering of our first research question on what problems remain with RL evaluation. In contrast, this section (and research question) concerns itself with the practical aspects of using the recommendations made in the guideline. Here, we conduct our algorithm comparison of DQN, A2C and PPO as a function of their performances on Cart Pole, Acrobot and Lunar Lander. From this analysis we discuss, in section 4.3, the effectiveness and shortcomings of our recommended guideline.

4.2.1 Sample Efficiency Curves

The first aspect of our algorithm evaluation which we consider is the presentation of the sample efficiency curves in Figures 21, 24, 25 and 26.[122] Figure 21 provides three plots which aggregate Figures 24, 25 and 26. From these one can consider the speed of training each of the algorithms, their sample efficiencies, the variability of each of their policies at each evaluated timestep, and their ability to converge to various levels of performance.

From the sample efficiency curves we can see that all three algorithms learn relatively quickly on Acrobot and converge to similar levels of performance with an average reward of approximately -80. On Cart Pole, only PPO converges to optimal performance. It achieves this with relatively high sample efficiency. DQN and A2C have highly variable results. Looking at the dis-aggregated results (Figure 24), A2C appears to perform much better than DQN as it appears to converge to optimal performance (an average reward of 500) in three of the five runs. Finally, in contrast to Cart Pole, DQN performs the best on Lunar Lander however none of the algorithms converge or achieve optimal levels of performance (an average reward above 200) on this task.[123]

These plots indicate that PPO may be our preferred candidate for the top ranked algorithm amongst the three being assessed. Our intuition is lead in this direction due to DQN performing particularly badly on Cart Pole, all algorithms performing similarly on Acrobot and due to PPO being relatively close to DQN on Lunar Lander. Furthermore, PPO performs similarly to A2C on Lunar Lander and Acrobot, but outperforms it on Cart Pole.[124]

We note that this presentation of the sample efficiency curves results from using the particular evaluation details for each task as presented in Tables 5, 6 and 7. Specifically, the plots show the 5 algorithm training runs up to 100 000 timesteps where algorithm performances are plotted every 10 000 timesteps as the average of 10 evaluation episodes of the policy (check-pointed at that timestep). Figure 21, which aggregates over algorithm runs, uses the inter-quartile range as the uncertainty measurement as recommended by Chan et al. [20].

[122] Figures 24, 25 and 26 appear in the appendix.

[123] Interestingly, A2C appears in the Lunar Lander task (in Figure 26) to have relatively unstable and high-variance performance scores as is demonstrated by the relatively large blue shaded region in this figure. Here, uncertainty is measured across the evaluation episodes of the check-pointed policies at every 10000 timesteps.

[124] Whilst looking at the training curves and performance scores in a naive way gives a strong intuition for which algorithm we prefer in this example experiment, in an experiment with more runs, more algorithms and more tasks, data aggregation (using MCDA) would provide a more beneficial use-case. This is since we may not be able to find an intuitively best algorithm in experiments with greater complexity. Experiments with algorithm results that are closer together may also benefit more from the recommended data aggregation.

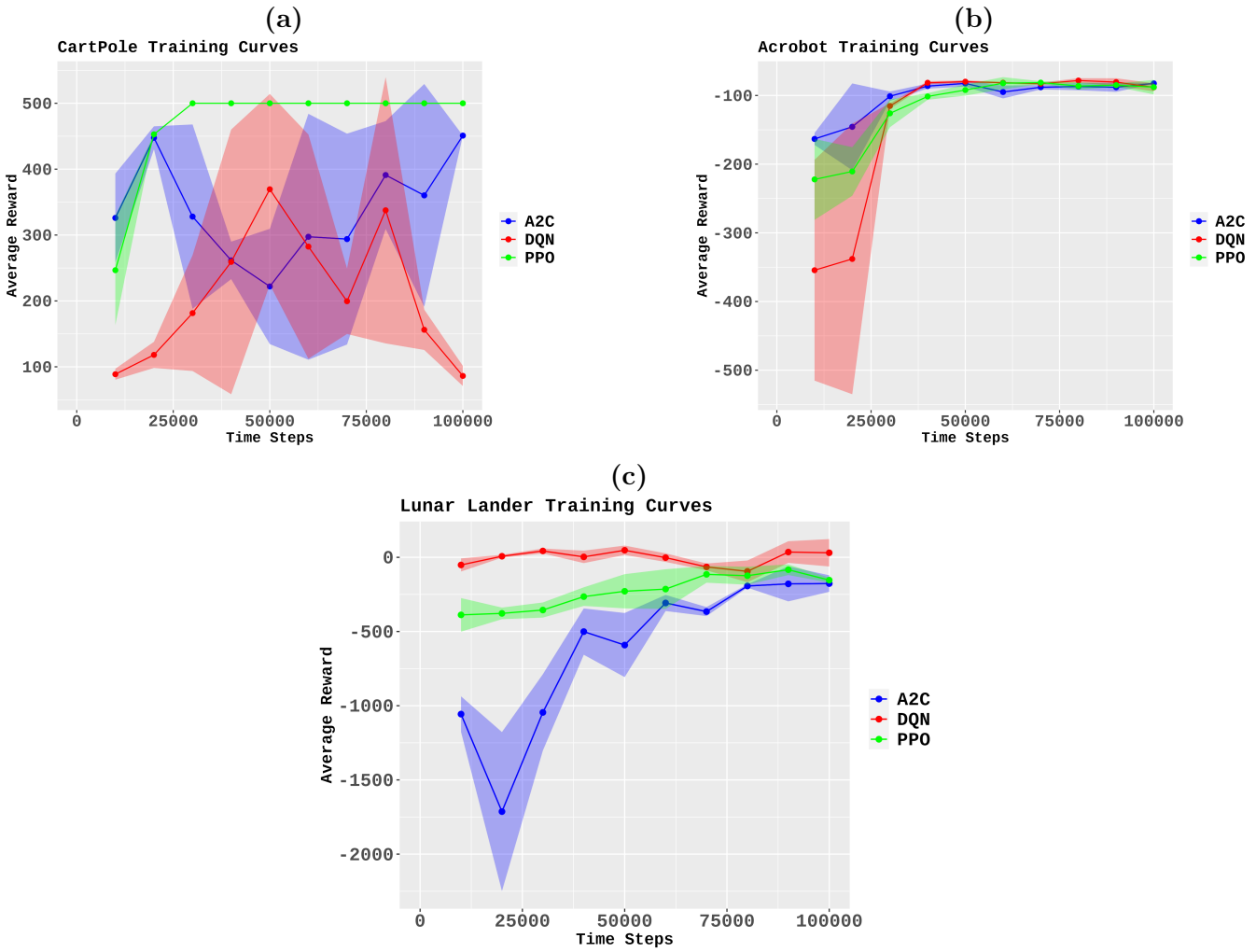

Figure 21: These plots show the training curves of A2C, DQN and PPO on Cart Pole (**a**), Acrobot (**b**) and Lunar Lander (**c**) respectively. The plots show the average reward of the three algorithms across the duration of training. The data points measured are aggregated across 5 training runs for each of the three algorithms. Each of the individual training runs can be seen in Figures 24, 25 and 26 in the appendix. The uncertainty intervals provided by the shaded regions are calculated using the inter-quartile range (IQR) which is recommended by Chan et al. [20].

4.2.2 Algorithm Performance Comparison

Using the algorithm runs presented above, we now compare the performances of the algorithms across tasks, using MCDA. The algorithm performance dataset, seen in Table 8, documents the dis-aggregated algorithm performance scores. These correspond to the mean rewards as achieved by the algorithms over 10 evaluation episodes, using the check-pointed policy at timestep 100 000. These we refer to as the *final metrics* [92]. This dataset provides performance scores $x_{n,j}(k)$ for algorithm k's runs, n, on the three RL tasks $j \in \Theta$. The dataset represents the conclusion of our data collection process as presented by our guideline.

With the above performance scores $x_{n,j}(k)$, we make use of our constructed example partial value functions (Equations 38, 39 and 40) combined with our example swing weights and our additive aggregation model (Equation 44), to aggregate across tasks and runs, to form individual performance scores for each algorithm $y(k)\ \forall k \in \mathcal{L}$. This yields the following results:[125]

[125] See our data aggregation notebook https://github.com/marlEvalDissertation/marlEvalDiss/blob/main/Guideline_Data_Aggregation.ipynb

Table 8: Mean rewards achieved at the end of training for three algorithms, on each of the five runs, on the three RL tasks

Algorithms	**DQN**	**A2C**	**PPO**
Cart Pole			
Run 1	28.7	254.7	500
Run 2	78.7	500	500
Run 3	91.9	500	500
Run 4	122.4	500	500
Run 5	109.2	500	500
Acrobot			
Run 1	-119.9	-82.3	-81
Run 2	-80.9	-85.8	-100.2
Run 3	-76.7	-81.9	-78.6
Run 4	-74.8	-83.2	-100.7
Run 5	-88.5	-79.1	-79
Lunar Lander			
Run 1	203.88	-233.84	-144.12
Run 2	220.56	-282.12	-241.09
Run 3	19.3	-30.34	-141.98
Run 4	192.55	-214.66	-176.32
Run 5	-483.13	-122.4	-64.03

$$\begin{aligned} y(DQN) &= 45.69 \\ y(A2C) &= 58.28 \\ y(PPO) &= 63.16 \end{aligned} \tag{45}$$

These results represent the conclusion of our data aggregation component of our analysis as recommended by our guideline. The results suggest that, based on our algorithm performances at 100 000 timesteps, for the given hyperparameters, for our three chosen tasks and for our constructed MCDA model, PPO outperforms A2C which outperforms DQN. While these results represent the conclusion of our data aggregation process, their presentation should not be viewed as the end of our analysis.

The MCDA process provides a means to reflect back on the decision-maker's preferences in terms of the performances of the algorithms on the selected tasks and the subjective judgemental information that they have provided. However, it is only an initial attempt at synthesising this information. The extent to which the MCDA process will be useful depends on the extent to which feedback can be provided [195].

The MCDA process should consider the model (and performance evaluation) as a catalyst for discussion [195]. A reflective process which considers the subjective judgemental information provided is also necessary. To do this one should consider the results as compared to the intuition we may have for our engineering problem. In the presence of competing partial value functions, constructed by different experts, we should reflect on the robustness of our performance assessment and perhaps revise the constructed value functions. In light of these results one should consider the inclusion of other tasks as criteria for the assessment of the algorithms. Lastly, a consideration of the uncertainty we have for the values of the weights may be analysed via a sensitivity analysis.

4.2.3 Sensitivity and Uncertainty Analysis

Despite us having conducted an aggregated performance evaluation over tasks it is useful for us to study the performance profiles of the algorithms in terms of their individual (weighted and normalised) scores on each task. Here, one must consider how an algorithm's aggregate value is made up, whether it performs well on all tasks or if its aggregate score is made up of very strong performances on some tasks which compensate for weak performances on others. Similarly we consider if any algorithms dominate, or are dominated by, the others. Figure 22 displays the performance profiles of our algorithms.[126]

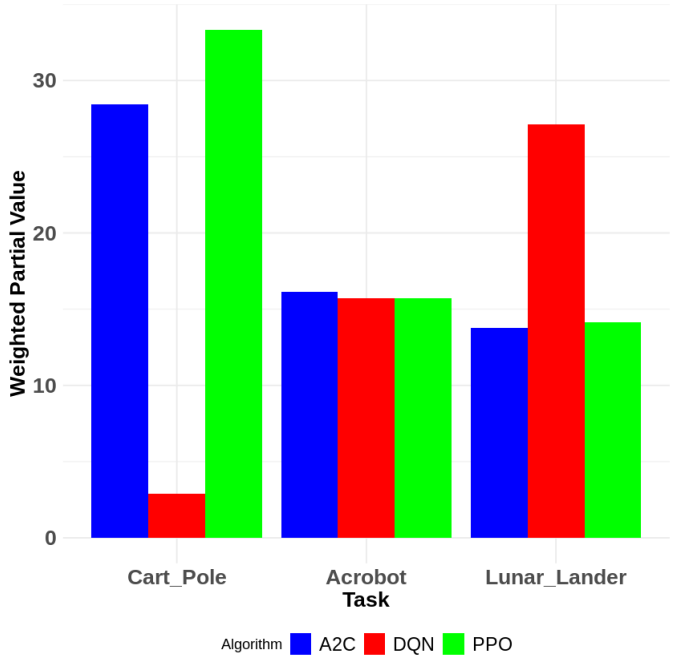

Figure 22: Weighted normalised scores of PPO, A2C and DQN on the RL tasks (Cart Pole, Acrobot and Lunar Lander).

Figure 22 shows the per-task (weighted and normalised) components of each of the algorithm's aggregate performance scores. As the scores in this plot are both weighted and normalised, the scores can be compared across tasks.[127] For instance the plot demonstrates that the scores of A2C and PPO on Lunar Lander are almost equivalent to the scores of the three algorithms on Acrobot. The figure demonstrates that the performance profiles of A2C and PPO in our evaluation are very similar. The only noteworthy difference between them occurs in Cart Pole.[128] DQN has a very different performance profile. Its score is mostly comprised from its relatively strong performance on Lunar Lander. Interestingly, as the rankings of the algorithms differ, especially when comparing Cart Pole and Lunar Lander, a different weighting of the tasks may produce a different result.

If we were to exclude A2C from the analysis (due to PPO's performance nearly dominating it) and if we were to exclude Acrobot (due to all algorithms performing similarly on this task) then we can perform a simplified sensitivity analysis of the weights of Lunar Lander and Cart Pole. This may aid us in determining the decision boundaries that lead to different overall algorithm rankings. Figure 23 demonstrates this.[129]

Recall our choice of weights having $q_{lunarlander} = 100$ and $q_{cartpole} = 75$. Figure 23 demonstrates this to be a robust choice of weights since small perturbations to this choice yield no change to the rank ordering of

[126] See our analysis notebook https://github.com/marlEvalDissertation/marlEvalDiss/blob/main/Results_Analysis.ipynb

[127] This means that one unit of score on any one task is equivalent to one unit of score on another task.

[128] One could interpret this result to be a demonstration that PPO dominates A2C.

[129] See our analysis notebook https://github.com/marlEvalDissertation/marlEvalDiss/blob/main/Results_Analysis.ipynb

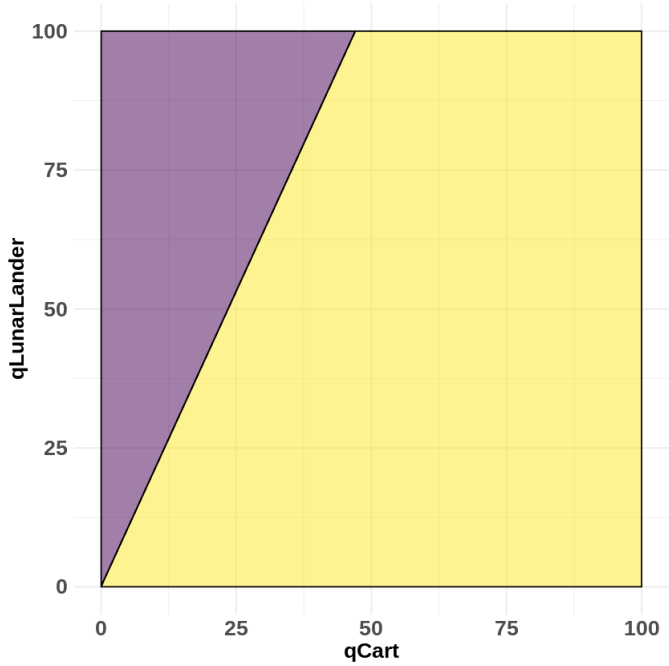

Figure 23: This plot demonstrates the sensitivity of the rankings of algorithms to the changes in weights. The purple region indicates the values of the weights of Cart Pole and Lunar Lander that lead to DQN outperforming PPO in terms of their aggregated MCDA performance scores. Similarly the yellow region indicates the values of these weights for which PPO outperforms DQN. Note that the Acrobot task and the A2C algorithm are both excluded from this analysis.

the algorithms. This is indicated as our choice of weights falls well within the yellow region of the figure. The yellow region represents the values of weights $q_{lunarlander}$ and $q_{cartpole}$ for which PPO outranks DQN. If we say that the decision boundary falls approximately on the line defined by $\frac{q_{cartpole}}{q_{lunarlander}} = 0.5$ then the weight for Lunar Lander would have to be more than double that of Cart Pole for there to be a change in the algorithm rankings.[130]

Finally, we investigate whether a change in algorithm rankings would occur if a min-max normalisation function were to be used alongside an aggregation function with a biased weighting [21] like the inter-quartile mean (IQM). We use the min-max normalisation function and IQM to aggregate over tasks and runs to produce aggregate performance scores for our algorithms. These are chosen due to the recommendations of Gorsane et al. [1] and Agarwal et al. [23]. This results in the following algorithm performance scores:[131]

$$\begin{aligned} y(DQN) &= 0.793 \\ y(A2C) &= 0.916 \\ y(PPO) &= 0.952 \end{aligned} \tag{46}$$

Interestingly, the use of the min-max normalisation function and IQM result in the same rank ordering of algorithms. One can anticipate PPO only slightly outperforming A2C due to their near equivalent performances on Acrobot and Lunar Lander and due to PPO slightly outperforming A2C on Cart Pole. Similarly, one could anticipate A2C and PPO outperforming DQN as they all perform similarly on Acrobot and Lunar Lander, with DQN's performance on Cart Pole being significantly worse.[132]

As the scores are normalised to $[0, 1]$, we note that PPO and A2C achieve near-perfect scores. This is

[130] This assumes that the partial value function constructions remain the same.

[131] See our analysis notebook https://github.com/marlEvalDissertation/marlEvalDiss/blob/main/Results_Analysis.ipynb

[132] See Figure 21.

due to us using the global minima and maxima found for the tasks' reward functions within the min-max normalisation function (as we did for our partial value functions). The choice to use local minima and maxima would, in contrast, directly result in lower values for the algorithm performance scores. The use of global minima in particular, combined with the linear scaling of task difficulty with reward levels achieved, means that the algorithm performances tend to 1 since they are far from initial values at the end of training.[133] This provides contrast with the scaling of normalised scores resulting from the construction of our partial value functions, which have aggregated performance scores far from 100.

This provides one resulting difference between our MCDA aggregation procedure and the use of a min-max function with an aggregation function like the IQM. Thus, we have successfully made use of our MCDA-enhanced guideline to perform an empirical evaluation of three deep RL algorithms. We now turn to the discussion of these results.

[133] and since we make use of the final metric as in Agarwal et al. [23].

4.3 Discussion: The Use of our Guideline

Having conducted a proof-of-concept test that performs an evaluation of deep RL algorithm performances using our guideline outlined in section 3.2, we now discuss its effectiveness (and shortcomings) in addressing our fourth research question: *Can the guideline be used to effectively evaluate deep RL algorithms and if so, how?* The results of section 4.2 offer evidence of the guideline's effectiveness. In this section, we discuss the processes of data collection and data aggregation as prescribed by the guideline, and in light of these results.

Our guideline was developed to address the problems with deep RL empirical algorithm performance evaluations. In answering our first research question, we identified two problems, both of which the guideline addresses (as discussed in section 3.3).[134] With the guideline, we aim to move towards a standardised protocol that goes from having a set of algorithms to having a set of robust aggregate performance scores for these algorithms. The algorithms can then be compared on the basis of these scores. As demonstrated by the algorithm performance scores $y(k)$ for algorithms $k \in \{\text{DQN, A2C, PPO}\}$ (seen in Equation 45) and as demonstrated by our data collection and data aggregation processes of the previous subsections, our evaluation process has achieved this.

The first step in compiling the aggregate scores $y(k)$ (after selecting the algorithms) was to select RL tasks by which to evaluate the algorithms. The guideline recommends selecting multiple tasks for evaluation as prescribed by the literature [1, 12, 16, 21, 23, 74]. Following this recommendation, we selected the RL tasks $\Theta = \{\text{Cart Pole, Acrobat, Lunar Lander}\}$. However, a limitation of this evaluation (as discussed in section 3.3) arises from the problem of which particular tasks to select. Consequently, the effectiveness of the guideline was constrained to adhering solely to the recommendation of selecting multiple tasks. We also recommend that the process of evaluation should catalyse discussion on the inclusion of more tasks.[135] [136] Furthermore, the process should be able to identify redundant tasks. For instance, in our evaluation of algorithms on the Acrobot task, performance scores were very close to one another (Figure 21 **(b)**). An iterative process that uses the evaluation procedure to provide feedback and catalyse modifications is recommended by the MCDA literature [195].

Given the selected tasks $j \in \Theta$ for evaluation, the data collection process, as recommended by the guideline, was effective in providing dis-aggregated performance scores $x_{n,j}(k)$ for each algorithm k, for runs $n \in \{1, .., 5\}$ on each task $j \in \Theta$. This is reflected by Figure 21 and Table 8. The guideline's aim was to provide a data collection process that provides robust measurements of algorithm performances across time as well as robust measurements of point estimates of algorithm performances after training. We achieved this by measuring the dis-aggregated performance scores as reflected by Figure 21 and Table 8. These were collected by following the recommendations to take the average of E evaluation episodes, conducted independently from training, every t^* timesteps, using the check-pointed policies at that timestep. These recommendations are consistent with the literature [14, 17, 19, 23]. Our use of the inter-quartile range (IQR), as recommended by Chan et al. [20], provided an effective measurement of uncertainty of the algorithm performances after (and during) training.[137] [138] This aided in providing robust dis-aggregated score measurements [1, 19, 23]. For our performance scores after training, we measured the final metrics [92]. We note that the variability of algorithm performances (across the training curves in Figure 21) provides a further indictment of the use of the absolute metric [92], which is used by Gorsane et al. [1].[139]

[134] These include the non-linear scaling of the reward scores achieved by algorithms with varying task difficulties, and the use of biased aggregation functions for weighting algorithms [21].

[135] For instance, in the case of an engineering problem where algorithms achieve similar performance scores $y(k)$ and only one algorithm is feasible for implementation, as the evaluator would like to distinguish between the algorithms, an extra RL task may be needed to provide further insight.

[136] The modification of the experiment through the inclusion/exclusion of tasks, in the case of a scientific test [24], would amount to a new experiment as an experimenter cannot invalidate results by changing the experiment.

[137] Again see Figure 21.

[138] Use of the reliability metrics by Chan et al. [20], for instance across a single run, may also be useful as discussed in section 2.2.

[139] and as discussed in section 2.2.3.1.

For our choice of evaluation parameters, we chose the evaluation frequency to be $t^* = 10000$. This value was chosen because it is the most commonly used choice of this parameter, as indicated by the meta-analysis conducted by Gorsane et al. [1] on cooperative MARL evaluations. We also chose to conduct $N = 5$ runs per algorithm per task due to Agarwal et al. [23] recommending conducting between $5 - 10$ runs per task for RL evaluations. Additionally, we chose to use $E = 10$ evaluation episodes, conducted independently from training, as this is a sufficient sample size for collecting data. Gorsane et al. [1] recommend using $E = 32$ due to commonality in the cooperative MARL literature. We note that our guideline leaves choices like these, of which specific experimental parameters to use, as an area of future work.

Another limitation with our experiments is that the experimental conclusions drawn from the produced scores are only pertinent to the chosen hyperparameters and specific experimental details as discussed in section 3.1.[140] This is no different than the conclusions drawn for most results that appear in the literature, with the noteworthy exception of the hyperparameter-agnostic algorithms that appear in Jordan et al. [21].

Following data collection, the data aggregation process (that was demonstrated through the above experiment) also provided evidence for the effectiveness of our guideline. We were able to construct example partial value functions for the chosen RL tasks based on how different levels of reward correspond to different levels of skill achievement for the algorithms (in a non-linear way). We were able to select weights based on our preferences for swings from worst to best rewards on the different RL tasks. Lastly, we were able to use an additive aggregation function to combine normalised performance scores with weights in a way that abides by the desired properties discussed.[141] This culminated in the aggregate algorithm performance scores $y(k)$ as seen in Equation 45.

To achieve the elicitation of these scores, we utilised preference modelling and data aggregation as prescribed by the MCDA process [25, 26, 191, 195]. The preference modelling process facilitated the construction of three example partial value functions (see Equations 38, 39 and 40). However, in this experiment, we constructed the partial value functions using a naive approach by studying the training curves of the algorithms. As noted in section 4.1, the true construction of such partial value functions requires specific value elicitation procedures (like the bisection and difference methods [192]), used iteratively with domain experts (preferably the authors of the RL tasks) [195]. Consistency checks with these experts are also crucial [194].[142] These procedures aid in providing a robust construction of the partial value functions [194].

In constructing the partial value functions, we made use of global minima and global maxima as reference points. We made this choice as we desire the construction of globally applicable partial value functions that are reusable across different evaluations. We note that it is possible to define local minima and maxima based on a set of algorithm performances that include human performances [23].[143] Many research papers also use a random agent's performance as a reference point [21]. While not done in this dissertation, we note that it is possible to select weights based on local reference points even if global reference points are used for the construction of partial value functions [25].

We also successfully made use of the bisection method and difference method by von Winterfeldt and Edwards [192]. We include these as examples of recommended elicitation methods for the construction of partial value functions. This is due to them being specifically designed for purpose [26, 195]. Equivalent methods can be found for constructing subjective utility functions in the field of subjective expected utility theory [194]. We note that it is possible for different experts to construct different partial value functions [25]. This may shed light on different perspectives available about the same task (or about algorithm behaviours on the same task). An iterative process of performing consistency checks and perhaps reaching a consensus may be useful to move from multiple to one prevailing partial value function [194].[144]

[140] and as appear in Tables 5, 6 and 7.
[141] See sections 2.2.3.2 and 2.4.4.
[142] These aspects are noted in sections 2.4.4 and 3.1.3.1.
[143] This relates to problems that demonstrate super-human performance [23].
[144] One such method of reaching consensus between multiple expert opinions is called the Delphi process [194].

We next looked at the swing weighting procedure as prescribed by our guideline. We again used a naive approach to select the example weights q_j culminating in the selection seen in Equation 41. A true determination of the weights would depend on the specific use-case at hand and how the evaluator views the trade-offs between increases in reward for the selected tasks. We note that in order to achieve a robust choice of weights, these should be selected using an iterative procedure of consistency checks as well as a sensitivity analysis [195].[145] Example consistency checks were performed in section 4.1.5.1, and an example sensitivity analysis can be seen in Figure 23. The sensitivity analysis in this study only looked at varying two weights, as the third task was found to be redundant. It is essential to expand this analysis to include the varying of all weights considered in an evaluation in order to assess their sensitivity to small changes. Using indifference points along with Equation 28 is also useful for assigning weights. This can be done by selecting (potentially hypothetical) pairwise indifference points between tasks. This is the most theoretically defensible way to assign weights [25].[146]

Swings weights account for both the relative importance of tasks with respect to one another and the extent to which the reward scales used discriminate between tasks [195]. Due to this and due to the effective use of partial value functions as normalisation functions which take into account the non-linear scaling of the reward scales with task difficulty, we were able to construct aggregate scores which are appropriately compatible with one another. These can be seen in Equation 45. The MCDA model used is consistent with the desired properties of having tasks which are measurable and preferentially independent, as well as value functions which are transitive, complete, and which transform average reward scores x to be on an interval scale of preferences $g(x)$ [25, 26, 195]. Furthermore, the MCDA model uses normalisation functions that consider both the location and scale of the tasks' reward functions, and which have scores which do not cluster in specific regions of $[0, 100]$ [21]. These thereby abide by the desired properties of normalisation functions as outlined in section 2.2.3.2.

In contrast, the scores in Equation 46 are calculated through the use of the min-max normalisation function and the inter-quartile mean (IQM) for aggregation. This procedure, as recommended by Gorsane et al. [1] and Agarwal et al. [23], provides incompatible performance scores due to the above properties not being met.[147] Furthermore, the use of global minima and maxima in the min-max normalisation function results in scores which are close to 1. This is due to algorithms achieving scores which are far from global minima after training. This encourages the use of local minima and maxima as reference points which may provide an element of subjectivity in these scores.

Figure 22 provides the weighted normalised scores of our three algorithms on the three tasks and thereby affords us a visualisation of the performance profiles of the algorithms [195]. Due to the aggregate scores abiding by the above properties, all scores reflected in this figure are compatible with one another. The figure allows us to view how each of the algorithm's aggregate scores are made up, and how each of the algorithms compare to one another on each of the tasks. This allows us to see if any algorithm dominates another algorithm, if any algorithm is an 'all rounder', or if an algorithm's aggregate score perhaps only benefits from its performance on one task. The use of such performance profiles provides important feedback to evaluators which can lead to the reconsideration of task weights [195].[148]

We again note that one major disadvantage with MCDA is that the process is subjective [194]. We note that this process provides but one example of an alternative to the use of min-max normalisation functions and to the use of aggregation functions like the mean, which are common in the literature [1, 23]. As we have demonstrated, our guideline addresses the flaws with these procedures and ensures a coherent process which is consistent with the desired properties. Furthermore, the subjective nature of MCDA may provide extra information to the evaluation process [25]. This may occur through the uncovering of differences between domain experts' constructed partial value functions and differences between their choices of swing

[145] As mentioned in sections 2.4.5 and 3.1.4.
[146] As mentioned in section 2.4.5.
[147] See a discussion on these in section 3.3.
[148] For instance, due to redundancies as we see with Acrobot in our example evaluation, or due to any particular task playing too large a role in the aggregate performance scores as determined by the evaluator.

weights. Including these aspects explicitly in the evaluation process ensures that information from any differences can add value to the evaluation process. The other main disadvantage of using our guideline with MCDA is that there is no measurement of uncertainty for the aggregate algorithm scores. This aspect is mitigated through the provision of per-task measurements of uncertainty, as we have effectively made use of the IQR function, as recommended by Chan et al. [20]. However, the lack of an aggregate uncertainty measurement or significance test is a major shortcoming of our guideline.[149]

Despite these shortcomings, we have provided evidence toward the effectiveness of using our guideline for improved deep RL empirical algorithm performance evaluations. In doing so, we have provided comments on *how* to make use of the guideline in order to perform evaluations which are more robust to judgemental errors.[150] The guideline meets the desired properties and improves upon the existing flaws with evaluations in the literature. It effectively performs data collection and data aggregation in line with the recommendations found in the literature. Crucially, the guideline does so whilst complying with our requirement to be computationally tractable [21]. Furthermore, the results consider the sensitivity to small changes in swing weights and consider the performance profiles of the algorithms across the tasks. These aspects provide feedback to assist the evaluator in considering potential changes to the model and potential re-evaluations. In the next section we conclude the dissertation by summarising our findings and offering paths to future research.

[149] As noted in section 3.3.

[150] This is accompanied by the example notebooks in https://github.com/marlEvalDissertation/marlEvalDiss/tree/main/

5 Conclusions

In this work, we aimed to address issues with deep RL empirical algorithm performance evaluations. To achieve this, we studied the components of and problems with these evaluations as identified in the literature. As a case study of an area of the field that has seen much recent research, we conducted an exploratory data analysis on the level of rigour of performance evaluations of deep cooperative MARL algorithms. Our findings motivated a search for alternative options to resolve the identified problems. MCDA was reviewed as a potential solution to these problems, especially regarding data aggregation. Therefore, we integrated MCDA with recommendations from the literature to formulate an improved guideline for deep RL evaluations. Finally, we provided a proof-of concept test of that guideline, which demonstrated how to use it. Through these efforts, we sought to answer the four research questions which appear in this dissertation's introduction.

To answer our first research question, *Are there specific problems with deep RL empirical algorithm performance evaluations, and why are these an issue?*, we conducted a literature review in section 2.2. Through this, we examined the following components of deep RL algorithm performance evaluations: task selection, uncertainty measurement, data collection and data aggregation. Various methods for performing these tasks were discussed, along with problems and proposed resolutions. In the introduction, it was hypothesised that we would find specific problems that remained with RL evaluation. Specifically, the issue of providing a normalisation function for algorithm k's performance score x_k was hypothesised to be problematic [21]. Our review provided evidence to this effect, as we demonstrated that the issues of normalising algorithms' performance scores and providing a non-biased aggregation procedure across algorithm tasks remained unresolved.

In answering our first research question, we found that commonly used normalisation functions failed to account for the non-linear scaling of task difficulty with levels of reward achieved, and the data aggregation functions used resulted in a biased weighting of tasks, when aggregating across tasks. Whilst Jordan et al. [21] offered one solution to these problems, they relied on performing thousands of runs using algorithms that did not tune hyperparameters. In our work, we sought to provide a computationally tractable solution that required only a few runs due to the high computational complexity of RL. Additionally, we considered comparing algorithms that had already undergone hyperparameter tuning.

In addressing our second research question, *What is the level of rigour in deep cooperative MARL empirical algorithm performance evaluations?*, we conducted an exploratory data analysis of a dataset documenting many recent cooperative MARL algorithm performance evaluations [1]. Evidence supporting the hypothesis of a poor level of rigour was drawn from the figures presented in section 2.3. It was observed that algorithm performances (for the same algorithm on the same task) exhibit wide variation across across research papers, a phenomenon prevalent across many tasks. Moreover, a select few tasks are recurrently chosen for evaluation (overfitting within the research field), and many evaluations utilise only one or two tasks to demonstrate performance (overfitting in individual research papers). Major inconsistencies were noted in the choice of aggregation function and in the choice of uncertainty measurement, with many research papers not reporting which they use. Additionally, many evaluations only conducted a single run per RL task (resulting in insignificant sample sizes). The problems with this recent subset of deep RL research motivated our continued effort for an improved RL evaluation guideline.

In order to resolve the two remaining problems with RL evaluations, identified through our first research question, we investigated the MCDA framework. This framed our third research question: *Can MCDA be used to benefit a guideline for coherent empirical algorithm performance evaluations, and if so, how?* As outlined in section 2.4, MCDA is a framework which responds to decision-problems that have multiple criteria for evaluation. As our problem resembles a decision-problem that looks to choose an algorithm from a set of algorithms, and as we have multiple tasks for evaluation, we framed deep RL algorithm performance evaluations as an MCDA problem. Notably, we found MCDA to specifically address the problems of normalising and aggregating data. It accomplishes this by constructing partial value functions

that accommodate the non-linear scaling of task difficulty with performance scores and by assigning weights that reflect the preference trade-offs between tasks, as well as the varying reward scales employed, thereby avoiding biases towards specific tasks over others when aggregating performance scores across tasks. Due to these two properties of the MCDA framework, we were able to demonstrate how MCDA is able to benefit RL evaluations. This responded to our third hypothesis which suggested that MCDA would benefit aggregation across tasks with different reward functions. In responding to this hypothesis, we attempted to construct a guideline for improved evaluations that made use of the insights from our literature review, along with MCDA. The constructed guideline can be found in section 3.2.

Notably, the guideline proposed using multiple tasks for evaluations, as per the recommendations in the literature [1, 12, 16, 21, 23, 74]. The guideline collects algorithm performance data using the *final metric*, the average performance of the last check-pointed policy at the end of training, evaluated over E evaluation episodes [14, 17, 20, 23, 92]. The guideline uses the inter-quartile range to provide an uncertainty measurement for the performance scores on each task [20]. As we seek a performance evaluation which is computationally tractable, we suggest using between five and ten runs per task for the evaluation [23]. To construct partial value functions, the proposed guideline recommends using the bisection and difference methods [192]. These are used in place of normalisation functions. For aggregation, we make use of the swing weighting process to weight the tasks, followed by an additive aggregation function [192]. This function combines the swing weights with the partial value function-normalised scores. The integration of partial value functions and swing weights into our guideline illustrates the effective incorporation of MCDA into RL performance evaluations.

Addressing our third research question illustrates the benefits that MCDA provides to RL evaluations. However, it is important to note that this represents only one suggestion for an improved guideline. MCDA requires the subjective judgement of trade-offs between various amounts of reward across (and between) tasks in order to construct partial value functions and assess swing-weights. MCDA also results in a lack of uncertainty measurement being available for the aggregate algorithm scores. However, the subjective element can be mitigated using the techniques advocated for in section 3.3, such as the use of domain experts in constructing the partial value functions.

Having demonstrated the benefit of using MCDA in theory, we then conducted a proof-of-concept test of our guideline to demonstrate its effectiveness and, therefore, respond to our fourth research question: *Given the aforementioned guideline, can this be used to effectively evaluate deep RL algorithms, and how?* Here, we implemented the guideline in practice by selecting three RL algorithms to be tested on three RL tasks. Data collection was conducted as per the recommendations in the guideline. Data aggregation was conducted using MCDA.[151] In this proof-of-concept test we were limited to forming only naive, example constructions of partial value functions and swing weights due to a lack of expert views available on the tasks chosen. However, we adhered to the procedures described by the bisection and difference methods (for eliciting user preferences to construct partial value functions), providing a proof-of-concept. This was accompanied by recommendations on how to conduct robust evaluations, such as performing consistency checks and conducting sensitivity analyses.

The experiments effectively demonstrated how to make use of the guideline. They yielded a comprehensive empirical algorithm performance evaluation of the three deep RL algorithms, while adhering to all guideline instructions. The results provided evidence for the effectiveness of using the guideline. We generated aggregate performance scores of the three algorithms and compared their performances. Additionally, we conducted a sensitivity analysis of the selected weights, demonstrating their robustness to small changes. Furthermore, we compared our results to those obtained using the min-max normalisation function in conjunction with an inter-quartile mean function for aggregating across tasks. We demonstrated that both approaches yielded equivalent rankings of algorithms, although the latter failed to provide a non-biased weighting of tasks and account for the non-linear scaling of reward levels with algorithm performance.

[151] as per the recommendations in our guideline.

As mentioned, this guideline offers one suggestion for an improved protocol for deep RL algorithm performance evaluations. Several aspects which have not been fully addressed are left to future work. Firstly, considerable room is left for determining which algorithms serve as optimal benchmarks. Of particular interest are the self-tuning algorithms used in Jordan et al. [21]. Research into algorithms that require minimal hyperparameter tuning may also prove beneficial [34]. Perhaps exploring penalty terms that penalise long hyperparameter tuning times and long training times should be investigated.

The consideration of which tasks to choose also remains an open question. A suite of many different types of tasks (such as sparse reward tasks) that test different aspects of algorithms (such as the ability to explore) may be desirable. In conjunction with the development of any new tasks, we recommend that corresponding partial value functions be developed. This would aid in the increased understanding and evaluation of the task. In such a case, it is worth investigating whether the construction of partial value functions in conjunction with new tasks leads to improved reward functions being constructed. Additionally, in constructing new RL tasks, a set of suggested evaluation parameters should be published. This would include, for instance, how long to train RL algorithms for.

One area of improvement for this study would be the inclusion of an investigation about the robust construction of partial value functions for RL tasks, using experts for construction. For instance, in the case of the development of a novel algorithm, an evaluation would be necessary. Testing the use of our guideline for this purpose would be useful not only to assess the novel algorithm but also to validate the use of the guideline and refine the process for achieving increased robustness when constructing partial value functions. A study testing the robustness of the partial value functions may be useful. This may involve multiple experts, using an iterative process that compares their constructed partial value functions. The value functions would be constructed using the mentioned elicitation methods, an iterative process, and consistency checks that test the experts' judgements with hypothetical scenarios about the scores on the tasks. Sensitivity analyses could be conducted by checking how small changes to the piece-wise functions' slopes affect the algorithm rankings. Using many tasks and comparing the similarities and differences between the constructed partial value functions across tasks may also be valuable in understanding the possibility (and ease) of constructing such value functions for different types of tasks.

Further improvements to this study would involve the development of an uncertainty measurement for the aggregated MCDA performance scores. This is necessary for robust performance assessments [1, 19, 23]. Lastly, it is important to note that computational improvements to RL may render the evaluation suggestions by Jordan et al. [21], to use their performance percentiles for normalisation and data aggregation, viable. As their suggestion includes an uncertainty measurement and as their procedure does not rely on subjective judgements, this would mitigate the remaining problems with our evaluation guideline. In closing, we emphasise that through the process of proposing an improved evaluation guideline for deep RL algorithm performance evaluations, our aim is to encourage the pursuit of improved and sustained dedication to achieving robust, rigorous standards in the field of machine learning research.

References

[1] R. Gorsane, O. Mahjoub, R. J. de Kock, R. Dubb, S. Singh, and A. Pretorius, "Towards a standardised performance evaluation protocol for cooperative MARL," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=am86qcwErJm

[2] D. Sculley, J. Snoek, A. B. Wiltschko, and A. Rahimi, "Winner's curse? on pace, progress, and empirical rigor," in *ICLR Workshop*, 2018. [Online]. Available: https://openreview.net/forum?id=rJWF0Fywf

[3] J. Z. Forde and M. Paganini, "The scientific method in the science of machine learning," in *ICLR 2019 Workshop on Debugging Machine Learning Models*, 2019.

[4] J. Foerster, "Deep multi-agent reinforcement learning," Ph.D. dissertation, University of Oxford, Magdalen College, University of Oxford, Oxford, United Kingdom, 2018. [Online]. Available: https://ora.ox.ac.uk/objects/uuid:a55621b3-53c0-4e1b-ad1c-92438b57ffa4/download_file?file_format=application%2Fpdf&safe_filename=DeepMARL.pdf&type_of_work=Thesis

[5] A. OroojlooyJadid and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," *Applied Intelligence*, vol. 53, p. 13677–13722, 2023.

[6] Y. Yang and J. Wang, "An overview of multi-agent reinforcement learning from game theoretical perspective," *arXiv preprint arXiv:2011.00583*, 2020.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, Cambridge, Massachusetts, 2018.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.

[9] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999. [Online]. Available: https://papers.nips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf

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

[11] R. S. Sutton, "Dyna, an Integrated Architecture for Learning, Planning, and Reacting," *SIGART Bull.*, vol. 2, no. 4, p. 160–163, Jul. 1991. [Online]. Available: https://doi.org/10.1145/122344.122377

[12] S. Whiteson, B. Tanner, M. E. Taylor, and P. Stone, "Protecting against evaluation overfitting in empirical reinforcement learning," in *2011 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. IEEE, 2011, pp. 120–127.

[13] R. Islam*, P. Henderson*, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," in *Reproducibility in Machine Learning Workshop (ICML)*, 2017.

[14] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *Journal of Artificial Intelligence Research*, vol. 61, pp. 523–562, 2018.

[15] P. Henderson, "Reproducibility and reusability in deep reinforcement learning," Master's thesis, McGill University, School of Computer Science, McGill University, Montreal, Canada, 2018. [Online]. Available: http://digitool.library.mcgill.ca/R/?func=dbin-jump-full&object_id=154594&local_base=GEN01-MCG02

[16] A. Zhang, N. Ballas, and J. Pineau, "A dissection of overfitting and generalization in continuous reinforcement learning," *arXiv preprint arXiv:1806.07937*, 2018.

[17] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the Thirty-Second AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[18] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "How many random seeds? statistical power analysis in deep reinforcement learning experiments," *arXiv preprint arXiv:1806.08295*, 2018.

[19] ——, "A Hitchhiker's Guide to Statistical Comparisons of Reinforcement Learning Algorithms," in *ICLR Worskhop on Reproducibility*, Nouvelle-Orléans, United States, May 2019. [Online]. Available: https://hal.science/hal-02369859

[20] S. C. Chan, S. Fishman, A. Korattikara, J. Canny, and S. Guadarrama, "Measuring the reliability of reinforcement learning algorithms," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SJlpYJBKvH

[21] S. Jordan, Y. Chandak, D. Cohen, M. Zhang, and P. Thomas, "Evaluating the performance of reinforcement learning algorithms," in *International Conference on Machine Learning*. PMLR, 2020, pp. 4962–4973.

[22] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep policy gradients: A case study on ppo and trpo," in *International Conference on Learning Representations*, 2020.

[23] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 29 304–29 320. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/f514cec81cb148559cf475e7426eed5e-Paper.pdf

[24] J. N. Hooker, "Testing heuristics: We have it all wrong," *Journal of heuristics*, vol. 1, pp. 33–42, 1995.

[25] L. Scott, "Legitimacy and decision making in developmental local government : participative MCDA in Stellenbosch," Ph.D. dissertation, Department of Mathematics and Applied Mathematics, University of Cape Town, Cape Town, South Africa, 2003. [Online]. Available: http://hdl.handle.net/11427/8589

[26] T. J. Stewart, "Decision modelling course pack (honours 2021), decision modelling, department of statistical sciences, university of cape town, cape town, south africa," February 2006.

[27] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[28] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–, Jan. 2016. [Online]. Available: http://dx.doi.org/10.1038/nature16961

[29] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1433–1440.

[30] D. Silver, "Introduction to Reinforcement Learning with David Silver Lecture 7: Policy Gradient Methods," University Lecture, 2015. [Online]. Available: https://www.davidsilver.uk/wp-content/uploads/2020/03/pg.pdf

[31] S. Gronauer and K. Dieopold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, vol. 55, pp. 1–49, 02 2022.

[32] B. Millidge, A. Tschantz, A. K. Seth, and C. L. Buckley, "Reinforcement learning as iterative and amortised inference," *arXiv preprint arXiv:2006.10524*, 2020.

[33] G. N. Tasse, S. James, and B. Rosman, "Generalisation in lifelong reinforcement learning through logical composition," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=ZOcX-eybqoL

[34] B. Millidge, "Applications of the Free Energy Principle to Machine Learning and Neuroscience," Ph.D. dissertation, University of Edinburgh, Institute for Adaptive and Neural Computation, School of Informatics, University of Edinburgh, Edinburgh, Scotland, 2021. [Online]. Available: https://era.ed.ac.uk/handle/1842/38235

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529—533, February 2015. [Online]. Available: http://europepmc.org/article/MED/25719670

[36] Y. Schraner, "Teacher-student curriculum learning for reinforcement learning," *arXiv preprint arXiv:2210.17368*, 2022.

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

[38] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019. [Online]. Available: https://arxiv.org/abs/1509.02971

[39] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[40] A. Tschantz, B. Millidge, A. K. Seth, and C. L. Buckley, "Control as Hybrid Inference," in *ICML 2020 Workshop on Theoretical Foundations of RL*, 2020.

[41] C. Formanek, A. Jeewa, J. Shock, and A. Pretorius, "Off-the-grid marl: Datasets and baselines for offline multi-agent reinforcement learning," in *Extended Abstract at the 2023 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS, 2023.

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

[43] T. Chu, S. Chinchali, and S. Katti, "Multi-agent reinforcement learning for networked system control," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=Syx7A3NFvH

[44] A. Pretorius, S. Cameron, E. Van Biljon, T. Makkink, S. Mawjee, J. du Plessis, J. Shock, A. Laterre, and K. Beguir, "A game-theoretic analysis of networked system control for common-pool resource management using multi-agent reinforcement learning," *Advances in neural information processing systems*, vol. 33, pp. 9983–9994, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/71c1806ca28b555c76650f52bb0d2810-Abstract.html

[45] M. Herrera, M. Pérez-Hernández, A. Kumar Parlikad, and J. Izquierdo, "Multi-agent systems and complex networks: Review and applications in systems engineering," *Processes*, vol. 8, no. 3, p. 312, 2020. [Online]. Available: https://www.mdpi.com/2227-9717/8/3/312

[46] A. Haydari and Y. Yılmaz, "Deep reinforcement learning for intelligent transportation systems: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 11–32, 2022.

[47] A. Nowé, P. Vrancx, and Y.-M. De Hauwere, "Game theory and multi-agent reinforcement learning," *Reinforcement Learning: State-of-the-Art*, pp. 441–470, 2012.

[48] L. S. Shapley, "Stochastic games*," *Proceedings of the National Academy of Sciences*, vol. 39, pp. 1095 – 1100, 1953.

[49] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of reinforcement learning and control*, pp. 321–384, 2021.

[50] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò, "Multi-agent reinforcement learning: A review of challenges and applications," *Applied Sciences*, vol. 11, no. 11, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/11/4948

[51] F. A. Oliehoek, "Decentralized POMDPs," in *Reinforcement Learning: State of the Art*, ser. Adaptation, Learning, and Optimization, M. Wiering and M. van Otterlo, Eds. Berlin, Germany: Springer Berlin Heidelberg, 2012, vol. 12, pp. 471–503.

[52] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ser. ICML'94. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, p. 157–163.

[53] J. F. Nash, "Equilibrium points in n-person games." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 36 1, pp. 48–9, 1950.

[54] L. Matignon, G. Laurent, and N. Fort-Piat, "Independent reinforcement learners in cooperative markov games: A survey regarding coordination problems," *The Knowledge Engineering Review*, vol. 27, pp. 1 – 31, 03 2012.

[55] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.

[56] M. Morris, T. D. Barrett, and A. Pretorius, "Universally expressive communication in multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=bBgNsEKUxmJ

[57] A. M. Hafiz and G. M. Bhat, "Deep q-network based multi-agent reinforcement learning with binary action agents," *arXiv preprint arXiv:2008.04109*, 2020.

[58] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, 2017. [Online]. Available: http://papers.nips.cc/paper/7217-multi-agent-actor-critic-for-mixed-cooperative-competitive-environments

[59] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*. Springer, 2017, pp. 66–83.

[60] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/c7635bfd99248a2cdef8249ef7bfbef4-Paper.pdf

[61] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 4295–4304. [Online]. Available: https://proceedings.mlr.press/v80/rashid18a.html

[62] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018, pp. 2974–2982.

[63] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative multi-agent games," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 611–24 624, 2022.

[64] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman, "Solving transition independent decentralized markov decision processes," *Journal of Artificial Intelligence Research*, vol. 22, pp. 423–455, 2004.

[65] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella, "Taming decentralized pomdps : Towards efficient policy computation for multiagent settings," *Proc. 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pp. 705–711, 2003. [Online]. Available: https://cir.nii.ac.jp/crid/1572824500736433408

[66] G. J. Laurent, L. Matignon, L. Fort-Piat *et al.*, "The world of independent learners is not markovian," *International Journal of Knowledge-based and Intelligent Engineering Systems*, vol. 15, no. 1, pp. 55–64, 2011.

[67] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. De Cote, "A survey of learning in multiagent environments: Dealing with non-stationarity," *arXiv preprint arXiv:1707.09183*, 2017.

[68] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. F. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, E. André, S. Koenig, M. Dastani, and G. Sukthankar, Eds. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018, pp. 2085–2087. [Online]. Available: http://dl.acm.org/citation.cfm?id=3238080

[69] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel, "A unified game-theoretic approach to multiagent reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017.

[70] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 5887–5896. [Online]. Available: https://proceedings.mlr.press/v97/son19a.html

[71] J. Castellini, F. A. Oliehoek, R. Savani, and S. Whiteson, "The representational capacity of action-value networks for multi-agent reinforcement learning," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 1862–1864.

[72] S. Sukhbaatar, a. szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/55b1927fdafef39c48e5b73b5d61ea60-Paper.pdf

[73] J. Su, S. C. Adams, and P. A. Beling, "Value-decomposition multi-agent actor-critics," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 11 352–11 360. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17353

[74] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A study on overfitting in deep reinforcement learning," *arXiv preprint arXiv:1804.06893*, 2018.

[75] J. Hu, S. Wang, S. Jiang, and M. Wang, "Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning," in *The Second Blogpost Track at ICLR 2023*, 2023. [Online]. Available: https://openreview.net/forum?id=Y8hONVbMSDj

[76] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," in *International conference on machine learning*. PMLR, 2016, pp. 1225–1234.

[77] J. Lin, R. Camoriano, and L. Rosasco, "Generalization properties and implicit regularization for multiple passes sgm," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 2340–2348. [Online]. Available: https://proceedings.mlr.press/v48/lina16.html

[78] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging procedural generation to benchmark reinforcement learning," in *International conference on machine learning*. PMLR, 2020, pp. 2048–2056.

[79] B. Efron, "Bootstrap methods: another look at the jackknife," in *Breakthroughs in statistics: Methodology and distribution*, S. Kotz and N. L. Johnson, Eds. New York, NY: Springer, 1992, pp. 569–593. [Online]. Available: https://doi.org/10.1007/978-1-4612-4380-9_41

[80] J. Z. Leibo, E. A. Dueñez-Guzman, A. Vezhnevets, J. P. Agapiou, P. Sunehag, R. Koster, J. Matyas, C. Beattie, I. Mordatch, and T. Graepel, "Scalable evaluation of multi-agent reinforcement learning with melting pot," in *International conference on machine learning*. PMLR, 2021, pp. 6187–6199.

[81] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. N. Foerster, and S. Whiteson, "The starcraft multi-agent challenge," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '19, vol. 4. Montreal, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 2186–2188.

[82] V. Amrhein, S. Greenland, and B. McShane, "Scientists rise up against statistical significance," *Nature*, vol. 567, no. 7748, pp. 305–307, 2019.

[83] D. Romer, "In praise of confidence intervals," in *AEA Papers and Proceedings*, vol. 110. American Economic Association 2014 Broadway, Suite 305, Nashville, TN 37203, 2020, pp. 55–60.

[84] R. Wasserstein, A. Schirm, and N. Lazar, "Moving to a world beyond "p < 0.05"," *American Statistician*, vol. 73, pp. 1–19, 03 2019.

[85] G. Gigerenzer, "Statistical rituals: The replication delusion and how we got there," *Advances in Methods and Practices in Psychological Science*, vol. 1, no. 2, pp. 198–218, 2018.

[86] S. Greenland, S. J. Senn, K. J. Rothman, J. B. Carlin, C. Poole, S. N. Goodman, and D. G. Altman, "Statistical tests, p values, confidence intervals, and power: a guide to misinterpretations," *European journal of epidemiology*, vol. 31, pp. 337–350, 2016.

[87] S. N. Goodman, D. Fanelli, and J. P. Ioannidis, "What does research reproducibility mean?" *Science Translational Medicine*, vol. 8, no. 341, p. 341ps12, 2016.

[88] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht, "Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. [Online]. Available: https://openreview.net/forum?id=cIrPX-Sn5n

[89] ——, "Comparative evaluation of cooperative multi-agent deep reinforcement learning algorithms," *arXiv preprint arXiv:2006.07869*, 2020.

[90] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[91] I. Saeed, A. C. Cullen, S. M. Erfani, and T. Alpcan, "Domain-aware multiagent reinforcement learning in navigation," in *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021*. IEEE, 2021, pp. 1–8. [Online]. Available: https://doi.org/10.1109/IJCNN52387.2021.9533975

[92] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms," in *International conference on machine learning*. PMLR, 2018, pp. 1039–1048.

[93] P. J. Fleming and J. J. Wallace, "How not to lie with statistics: The correct way to summarize benchmark results," *Commun. ACM*, vol. 29, no. 3, p. 218–221, mar 1986. [Online]. Available: https://doi.org/10.1145/5666.5673

[94] W. M. Dabney, "Adaptive step-sizes for reinforcement learning," Ph.D. dissertation, University of Massachusetts Amherst, Graduate School of the University of Massachusetts Amherst, Massachusetts, USA, 2014. [Online]. Available: https://scholarworks.umass.edu/dissertations_2/173

[95] S. Omidshafiei, C. Papadimitriou, G. Piliouras, K. Tuyls, M. Rowland, J.-B. Lespiau, W. M. Czarnecki, M. Lanctot, J. Perolat, and R. Munos, "α-rank: Multi-agent evaluation by evolution," *Scientific reports*, vol. 9, no. 1, p. 9937, 2019.

[96] M. Rowland, S. Omidshafiei, K. Tuyls, J. Perolat, M. Valko, G. Piliouras, and R. Munos, "Multiagent evaluation under incomplete information," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[97] S. L. Salzberg, "On comparing classifiers: Pitfalls to avoid and a recommended approach," *Data Mining and Knowledge Discovery*, vol. 1, pp. 317–328, 1997.

[98] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Annals of Mathematical Statistics*, vol. 18, pp. 50–60, 1947.

[99] Łukasz Kaiser, M. Babaeizadeh, P. Miłos, B. Osiński, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski, "Model based reinforcement learning for atari," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=S1xCPJHtDB

[100] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study," in *ICLR 2021 - Ninth International Conference on Learning Representations*, Vienna / Virtual, Austria, 2021. [Online]. Available: https://inria.hal.science/hal-03162554

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

[102] G. Tesauro *et al.*, "Temporal difference learning and td-gammon," *Communications of the Association for Computing Machinery*, vol. 38, no. 3, pp. 58–68, 1995.

[103] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, 2017.

[104] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aar6404

[105] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[106] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aam6960

[107] N. Brown and T. Sandholm, "Superhuman ai for heads-up no-limit poker: Libratus beats top professionals," *Science*, vol. 359, no. 6374, pp. 418–424, 2018. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aao1733

[108] ——, "Superhuman ai for multiplayer poker," *Science*, vol. 365, no. 6456, pp. 885–890, 2019. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aay2400

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

[110] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel, "Human-level performance in 3d multiplayer games with population-based reinforcement learning," *Science*, vol. 364, no. 6443, pp. 859–865, 2019. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aau6249

[111] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SkxpxJBKwS

[112] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical programming*, vol. 91, pp. 201–213, 2002.

[113] R. Dror, S. Shlomov, and R. Reichart, "Deep dominance-how to properly compare deep neural models," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 2773–2785.

[114] H. Levy, "Stochastic dominance and expected utility: Survey and analysis," *Management Science*, vol. 38, pp. 555–593, 04 1992.

[115] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. C. Courville, and P. Bachman, "Data-efficient reinforcement learning with self-predictive representations," in *International Conference on Learning Representations*, 2020.

[116] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, pp. 156 – 172, 04 2008.

[117] C. Zhu, M. Dastani, and S. Wang, "A survey of multi-agent reinforcement learning with communication," *arXiv preprint arXiv:2203.08975*, 2022.

[118] A. Singh, T. Jain, and S. Sukhbaatar, "Learning when to communicate at scale in multiagent cooperative and competitive tasks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rye7knCqK7

[119] J. Jiang, C. Dun, T. Huang, and Z. Lu, "Graph convolutional reinforcement learning," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HkxdQkSYDB

[120] A. Das, T. Gervet, J. Romoff, D. Batra, D. Parikh, M. Rabbat, and J. Pineau, "TarMAC: Targeted multi-agent communication," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 1538–1546. [Online]. Available: https://proceedings.mlr.press/v97/das19a.html

[121] A. Malysheva, D. Kudenko, and A. Shpilman, "Magnet: Multi-agent graph network for deep multi-agent reinforcement learning," in *2019 XVI International Symposium "Problems of Redundancy in Information and Control Systems" (REDUNDANCY)*, 2019, pp. 171–176.

[122] Y. Liu, W. Wang, Y. Hu, J. Hao, X. Chen, and Y. Gao, "Multi-agent game abstraction via graph attention neural network," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, pp. 7211–7218, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/6211

[123] A. Agarwal, S. Kumar, K. P. Sycara, and M. Lewis, "Learning transferable cooperative behavior in multi-agent teams," in *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 1741–1743.

[124] S. Q. Zhang, Q. Zhang, and J. Lin, "Efficient communication in multi-agent reinforcement learning via variance based control," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/14cfdb59b5bda1fc245aadae15b1984a-Paper.pdf

[125] ——, "Succinct and robust multi-agent communication with temporal message control," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 17 271–17 282. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/c82b013313066e0702d58dc70db033ca-Paper.pdf

[126] Y. Niu, R. Paleja, and M. Gombolay, "Multi-agent graph-attention communication and teaming," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '21. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2021, p. 964–973.

[127] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[128] F. Christianos, L. Schäfer, and S. Albrecht, "Shared experience actor-critic for multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 10 707–10 717. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/7967cc8e3ab559e68cc944c44b1cf3e8-Paper.pdf

[129] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[130] Y. Du, L. Han, M. Fang, J. Liu, T. Dai, and D. Tao, "Liir: Learning individual intrinsic reward in multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/07a9d3fed4c5ea6b17e80258dee231fa-Paper.pdf

[131] A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson, "Maven: Multi-agent variational exploration," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/f816dc0acface7498e10496222e9db10-Paper.pdf

[132] C. Schroeder de Witt, J. Foerster, G. Farquhar, P. Torr, W. Boehmer, and S. Whiteson, "Multi-agent common knowledge reinforcement learning," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/f968fdc88852a4a3a27a81fe3f57bfc5-Paper.pdf

[133] N. Carion, N. Usunier, G. Synnaeve, and A. Lazaric, "A structured prediction approach for generalization in cooperative multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/3c3c139bd8467c1587a41081ad78045e-Paper.pdf

[134] J. Xu, F. Zhong, and Y. Wang, "Learning multi-agent coordination for enhancing target coverage in directional sensor networks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 10 053–10 064. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/7250eb93b3c18cc9daa29cf58af7a004-Paper.pdf

[135] J. Roy, P. Barde, F. Harvey, D. Nowrouzezahrai, and C. Pal, "Promoting coordination through policy regularization in multi-agent deep reinforcement learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 15 774–15 785. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/b628386c9b92481fab68fbf284bd6a64-Paper.pdf

[136] Z. Ding, T. Huang, and Z. Lu, "Learning individually inferred communication for multi-agent cooperation," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 22 069–22 079. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/fb2fcd534b0ff3bbed73cc51df620323-Paper.pdf

[137] M. Zhou, Z. Liu, P. Sui, Y. Li, and Y. Y. Chung, "Learning implicit credit assignment for cooperative multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 11 853–11 864. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/8977ecbb8cb82d77fb091c7a7f186163-Paper.pdf

[138] J. Chen, Y. Zhang, Y. Xu, H. Ma, H. Yang, J. Song, Y. Wang, and Y. Wu, "Variational automatic curriculum learning for sparse-reward cooperative multi-agent problems," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 9681–9693. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/503e7dbbd6217b9a591f3322f39b5a6c-Paper.pdf

[139] M. Chen, Y. Li, E. Wang, Z. Yang, Z. Wang, and T. Zhao, "Pessimism meets invariance: Provably efficient offline mean-field multi-agent rl," in *Advances in Neural Information*

Processing Systems, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 17 913–17 926. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/9559fc73b13fa721a816958488a5b449-Paper.pdf

[140] J. Wang, Z. Ren, B. Han, J. Ye, and C. Zhang, "Towards understanding cooperative multi-agent q-learning with value factorization," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 29 142–29 155. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/f3f1fa1e4348bfbebdeee8c80a04c3b9-Paper.pdf

[141] K. M. Lee, S. G. Subramanian, and M. Crowley, "Investigation of independent reinforcement learning algorithms in multi-agent environments," in *Deep RL Workshop NeurIPS 2021*, 2021. [Online]. Available: https://openreview.net/forum?id=8MkKGZ2AlmJ

[142] L. Chenghao, T. Wang, C. Wu, Q. Zhao, J. Yang, and C. Zhang, "Celebrating diversity in shared multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 3991–4002. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/20aee3a5f4643755a79ee5f6a73050ac-Paper.pdf

[143] T. Rashid, G. Farquhar, B. Peng, and S. Whiteson, "Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 10 199–10 210. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/73a427badebe0e32caa2e1fc7530b7f3-Paper.pdf

[144] L. Zheng, J. Chen, J. Wang, J. He, Y. Hu, Y. Chen, C. Fan, Y. Gao, and C. Zhang, "Episodic multi-agent reinforcement learning with curiosity-driven exploration," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 3757–3769. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/1e8ca836c962598551882e689265c1c5-Paper.pdf

[145] L. Pan, T. Rashid, B. Peng, L. Huang, and S. Whiteson, "Regularized softmax deep multi-agent q-learning," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 1365–1377. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/0a113ef6b61820daa5611c870ed8d5ee-Paper.pdf

[146] J. G. Kuba, M. Wen, L. Meng, s. gu, H. Zhang, D. Mguni, J. Wang, and Y. Yang, "Settling the variance of multi-agent policy gradients," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 13 458–13 470. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/6fe6a8a6e6cb710584efc4af0c34ce50-Paper.pdf

[147] B. Peng, T. Rashid, C. Schroeder de Witt, P.-A. Kamienny, P. Torr, W. Boehmer, and S. Whiteson, "Facmac: Factored multi-agent centralised policy gradients," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 12 208–12 221. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/65b9eea6e1cc6bb9f0cd2a47751a186f-Paper.pdf

[148] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 1146–1155. [Online]. Available: https://proceedings.mlr.press/v70/foerster17b.html

[149] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 2961–2970. [Online]. Available: https://proceedings.mlr.press/v97/iqbal19a.html

[150] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2681–2690.

[151] N. Jaques, A. Lazaridou, E. Hughes, C. Gulcehre, P. Ortega, D. Strouse, J. Z. Leibo, and N. De Freitas, "Social influence as intrinsic motivation for multi-agent deep reinforcement learning," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 3040–3049. [Online]. Available: https://proceedings.mlr.press/v97/jaques19a.html

[152] W. Boehmer, V. Kurin, and S. Whiteson, "Deep coordination graphs," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 980–991. [Online]. Available: https://proceedings.mlr.press/v119/boehmer20a.html

[153] W.-F. Sun, C.-K. Lee, and C.-Y. Lee, "Dfac framework: Factorizing the value function via quantile mixture for multi-agent distributional q-learning," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 9945–9954. [Online]. Available: https://proceedings.mlr.press/v139/sun21c.html

[154] F. Christianos, G. Papoudakis, M. A. Rahman, and S. V. Albrecht, "Scaling multi-agent reinforcement learning with selective parameter sharing," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 1989–1998. [Online]. Available: https://proceedings.mlr.press/v139/christianos21a.html

[155] J. Jiang and Z. Lu, "The emergence of individuality," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 4992–5001. [Online]. Available: https://proceedings.mlr.press/v139/jiang21g.html

[156] I.-J. Liu, U. Jain, R. A. Yeh, and A. Schwing, "Cooperative exploration for multi-agent deep reinforcement learning," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 6826–6836. [Online]. Available: https://proceedings.mlr.press/v139/liu21j.html

[157] Z. Xu, D. Li, Y. Bai, and G. Fan, "MMD-MIX: value function factorisation with maximum mean discrepancy for cooperative multi-agent reinforcement learning," in *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021*. IEEE, 2021, pp. 1–7. [Online]. Available: https://doi.org/10.1109/IJCNN52387.2021.9533636

[158] Y. Xiao, X. Lyu, and C. Amato, "Local advantage actor-critic for robust multi-agent deep reinforcement learning," in *2021 International symposium on multi-robot and multi-agent systems (MRS)*. IEEE, 2021, pp. 155–163.

[159] I.-J. Liu, R. A. Yeh, and A. G. Schwing, "Pic: Permutation invariant critic for multi-agent deep reinforcement learning," in *Proceedings of the Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, 30 Oct–01 Nov 2020, pp. 590–602. [Online]. Available: https://proceedings.mlr.press/v100/liu20a.html

[160] E. Marchesini and A. Farinelli, "Centralizing state-values in dueling networks for multi-robot reinforcement learning mapless navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021*. IEEE, 2021, pp. 4583–4588. [Online]. Available: https://doi.org/10.1109/IROS51168.2021.9636349

[161] T. Wang, J. Wang, Y. Wu, and C. Zhang, "Influence-based multi-agent exploration," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=BJgy96EYvr

[162] W. Wang, T. Yang, Y. Liu, J. Hao, X. Hao, Y. Hu, Y. Chen, C. Fan, and Y. Gao, "Action semantics network: Considering the effects of actions in multiagent systems," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=ryg48p4tPH

[163] T. Wang, J. Wang, C. Zheng, and C. Zhang, "Learning nearly decomposable value functions via communication minimization," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HJx-3grYDB

[164] Q. Long*, Z. Zhou*, A. Gupta, F. Fang, Y. Wu†, and X. Wang†, "Evolutionary population curriculum for scaling multi-agent reinforcement learning," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SJxbHkrKDH

[165] H. Hu and J. N. Foerster, "Simplified action decoder for deep multi-agent reinforcement learning," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1xm3RVtwB

[166] T. Wang, T. Gupta, A. Mahajan, B. Peng, S. Whiteson, and C. Zhang, "{RODE}: Learning roles to decompose multi-agent tasks," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=TTUVg6vkNjK

[167] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang, "{QPLEX}: Duplex dueling multi-agent q-learning," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=Rcmk0xxIQV

[168] D. H. Mguni, T. Jafferjee, J. Wang, N. Perez-Nieves, O. Slumbers, F. Tong, Y. Li, J. Zhu, Y. Yang, and J. Wang, "LIGS: Learnable intrinsic-reward generation selection for multi-agent learning," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=CpTuR2ECuW

[169] Y. Wang, fangwei zhong, J. Xu, and Y. Wang, "Tom2c: Target-oriented multi-agent communication and cooperation with theory of mind," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=2t7CkQXNpuq

[170] J. G. Kuba, R. Chen, M. Wen, Y. Wen, F. Sun, J. Wang, and Y. Yang, "Trust region policy optimisation in multi-agent reinforcement learning," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=EcGGFkNTxdJ

[171] S. A. Stavroulakis and B. Sengupta, "Reinforcement learning for location-aware warehouse scheduling," in *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*, 2022. [Online]. Available: https://openreview.net/forum?id=Bt-gaVaVJ-9

[172] A. Castagna and I. Dusparic, "Multi-agent transfer learning in reinforcement learning-based ride-sharing systems," in *Proceedings of the 14th International Conference on Agents and Artificial Intelligence, ICAART 2022, Volume 2, Online Streaming, February 3-5, 2022*, A. P. Rocha, L. Steels, and H. J. van den Herik, Eds. SCITEPRESS, 2022, pp. 120–130. [Online]. Available: https://doi.org/10.5220/0010785200003116

[173] E. Wei, D. Wicke, D. Freelan, and S. Luke, "Multiagent soft q-learning," in *2018 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 26-28, 2018*. AAAI Press, 2018. [Online]. Available: https://aaai.org/ocs/index.php/SSS/SSS18/paper/view/17508

[174] J. Wang, Y. Zhang, T.-K. Kim, and Y. Gu, "Shapley q-value: A local reward approach to solve global reward games," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 7285–7292.

[175] C. Wen, X. Yao, Y. Wang, and X. Tan, "Smix(λ): Enhancing centralized value functions for cooperative multi-agent reinforcement learning," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 7301–7308. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/6223

[176] P. Leroy, D. Ernst, P. Geurts, G. Louppe, J. Pisane, and M. Sabatelli, "QVMix and QVMix-Max: Extending the Deep Quality-Value Family of Algorithms to Cooperative Multi-Agent Reinforcement Learning," in *Proceedings of the AAAI-21 Workshop on Reinforcement Learning in Games*, 2021. [Online]. Available: https://arxiv.org/abs/2012.12062

[177] B. Guresti and N. K. Ure, "Evaluating generalization and transfer capacity of multi-agent reinforcement learning across variable number of agents," in *2021 AAAI Challenges and Opportunities for Multi-Agent Reinforcement Learning Spring Symposium, Stanford University, Palo Alto, California, USA*. AAAI Press, 2021.

[178] L. Yuan, J. Wang, F. Zhang, C. Wang, Z. Zhang, Y. Yu, and C. Zhang, "Multi-agent incentive communication via decentralized teammate modeling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 9, 2022, pp. 9466–9474.

[179] X. Lyu, A. Baisero, Y. Xiao, and C. Amato, "A deeper understanding of state-based critics in multi-agent reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 36, no. 9, 2022, pp. 9396–9404.

[180] H. Mao, Z. Zhang, Z. Xiao, and Z. Gong, "Modelling the dynamic joint policy of teammates with attention multi-agent DDPG," in *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 1108–1116.

[181] J. Ma and F. Wu, "Feudal multi-agent deep reinforcement learning for traffic signal control," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '20. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2020, p. 816–824.

[182] S. Li, J. K. Gupta, P. Morales, R. Allen, and M. J. Kochenderfer, "Deep implicit coordination graphs for multi-agent reinforcement learning," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '21. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2021, p. 764–772.

[183] M. Zawalski, B. Osinski, H. Michalewski, and P. Milos, "Off-policy correction for multi-agent reinforcement learning," in *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2022, pp. 1774–1776.

[184] R. Avalos, M. Reymond, A. Nowé, and D. M. Roijers, "Local advantage networks for cooperative multi-agent reinforcement learning," in *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2022, pp. 1524–1526.

[185] J.-J. Wang, Y.-Y. Jing, C.-F. Zhang, and J.-H. Zhao, "Review on multi-criteria decision analysis aid in sustainable energy decision-making," *Renewable and Sustainable Energy Reviews*, vol. 13, no. 9, pp. 2263–2278, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032109001166

[186] G. Montibeller and A. Franco, *Multi-Criteria Decision Analysis for Strategic Decision Making*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 25–48. [Online]. Available: https://doi.org/10.1007/978-3-540-92828-7_2

[187] G. Mendoza and H. Martins, "Multi-criteria decision analysis in natural resource management: A critical review of methods and new modelling paradigms," *Forest Ecology and Management*, vol. 230, no. 1, pp. 1–22, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378112706002258

[188] I. Ivlev, J. Vacek, and P. Kneppo, "Multi-criteria decision analysis for supporting the selection of medical devices under uncertainty," *European Journal of Operational Research*, vol. 247, no. 1, pp. 216–228, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221715004877

[189] A. S. Yalcin, H. S. Kilic, and D. Delen, "The use of multi-criteria decision-making methods in business analytics: A comprehensive literature review," *Technological Forecasting and Social Change*, vol. 174, p. 121193, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0040162521006260

[190] B. Roy and M. R. McCord, *Multicriteria methodology for decision aiding*, 1st ed., ser. Nonconvex optimization and its applications ; Volume 12. Dordrecht: Springer-Science+Business Media, B.V., 1996.

[191] V. Belton and T. Stewart, *Multiple criteria decision analysis: an integrated approach*. Springer Science & Business Media, 2002.

[192] D. von Winterfeldt and W. Edwards, "Decision analysis and behavioral research," in *Cambridge University Press*, 1986.

[193] R. Keeney, H. Raiffa, and D. Rajala, "Decisions with multiple objectives: Preferences and value trade-offs," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 9, pp. 403 – 403, 08 1979.

[194] T. J. Stewart, "Bayesian decision modelling course pack (sta5061z bayesian decision modelling masters 2021) department of statistical sciences, university of cape town, cape town, south africa," 2021.

[195] T. Stewart, "Value function methods for discrete choice," 2023, unpublished.

[196] A. P. Wierzbicki, "Reference point approaches and objective ranking," in *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[197] T. J. Stewart, "Use of piecewise linear value functions in interactive multicriteria decision support: a monte carlo study," *Management Science*, vol. 39, pp. 1369–1381, 1993. [Online]. Available: https://api.semanticscholar.org/CorpusID:122786825

[198] ——, "Robustness of additive value function methods in mcdm," *Journal of Multi-criteria Decision Analysis*, vol. 5, pp. 301–309, 1996. [Online]. Available: https://api.semanticscholar.org/CorpusID:123135055

[199] S. R. Watson and D. M. Buede, *Decision synthesis: The principles and practice of decision analysis*. Cambridge University Press, 1987.

[200] C. A. B. e Costa and J.-C. Vansnick, "Applications of the macbeth approach in the framework of an additive aggregation model," *Journal of Multi-criteria Decision Analysis*, vol. 6, pp. 107–114, 1997.

[201] G. E. Box and N. R. Draper, *Empirical model-building and response surfaces*. John Wiley & Sons, 1987.

[202] A. Raffin, "Rl baselines3 zoo," https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[203] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025

[204] B. Millidge, "Deep active inference as variational policy gradients," *Journal of Mathematical Psychology*, vol. 96, p. 102348, 2020.

[205] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.

[206] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems*, D. Touretzky, M. Mozer, and M. Hasselmo, Eds., vol. 8. MIT Press, 1995. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1995/file/8f1d43620bc6bb580df6e80b0dc05c48-Paper.pdf

[207] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

Appendix

Sample Efficiency Curves

Figures 24, 25 and 26 show the sample efficiency curves of algorithms DQN, A2C and PPO on Cart Pole (Figure 24), Acrobot (Figure 25) and Lunar Lander (Figure 26). The presentation of the sample efficiency curves results from using the particular evaluation details for each task as presented in Tables 5, 6 and 7 in section 4.1. Specifically, the plots show the 5 algorithm training runs up to 100 000 timesteps where algorithm performances are plotted every 10 000 timesteps as the average of 10 evaluation episodes of the policy (check-pointed at that timestep). Figure 21 (section 4.2) aggregates over the 5 algorithm runs.

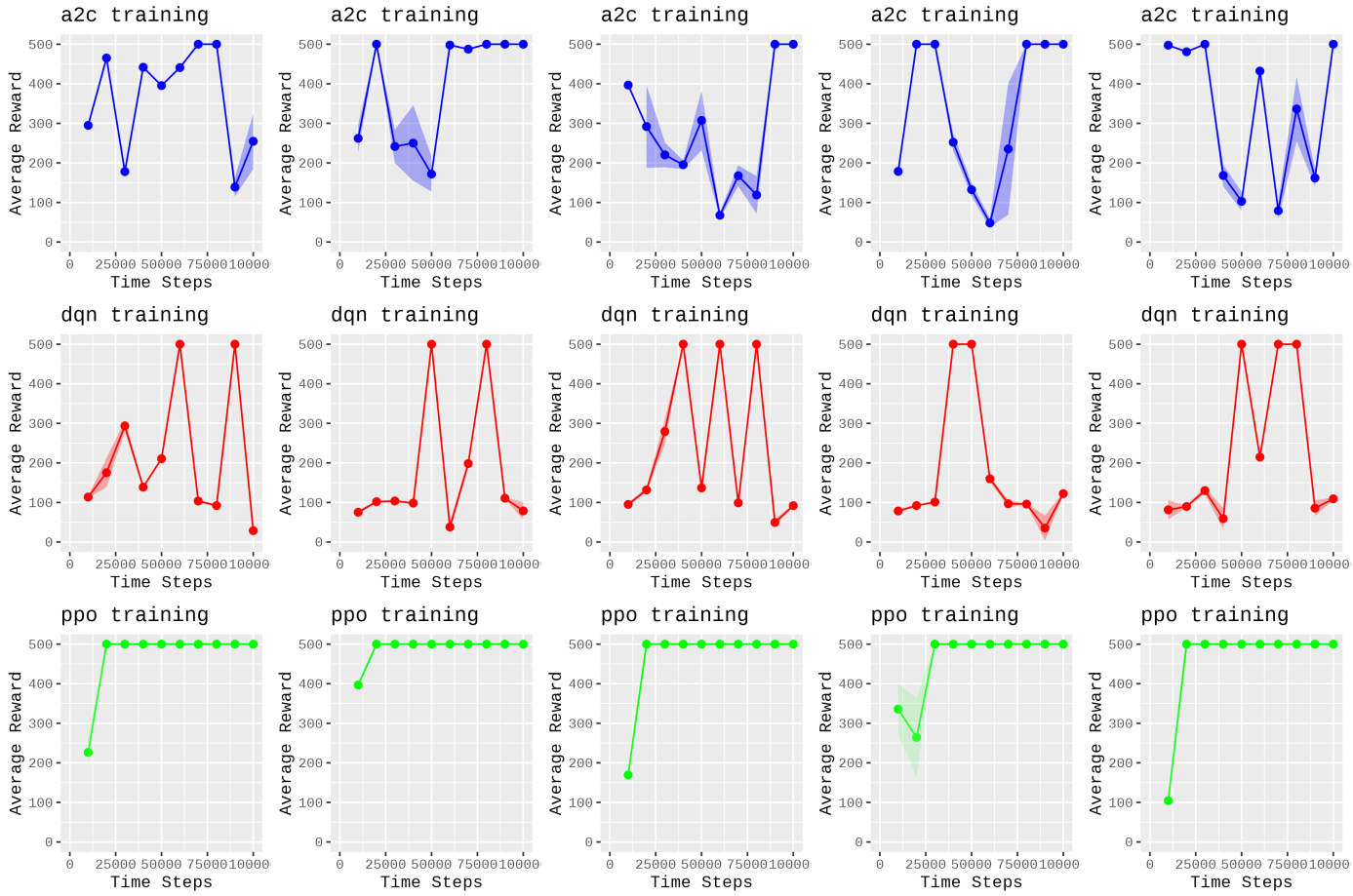

Figure 24: These plots show the individual training runs of the A2C (blue), DQN (red) and PPO (green) algorithms on the **Cart Pole** RL task. Each algorithm undergoes five individual training runs, each of which plotted separately. Data points are calculated by taking the mean rewards obtained from the check-pointed policy at the respective timestep, which is evaluated over $E = 10$ evaluation episodes.

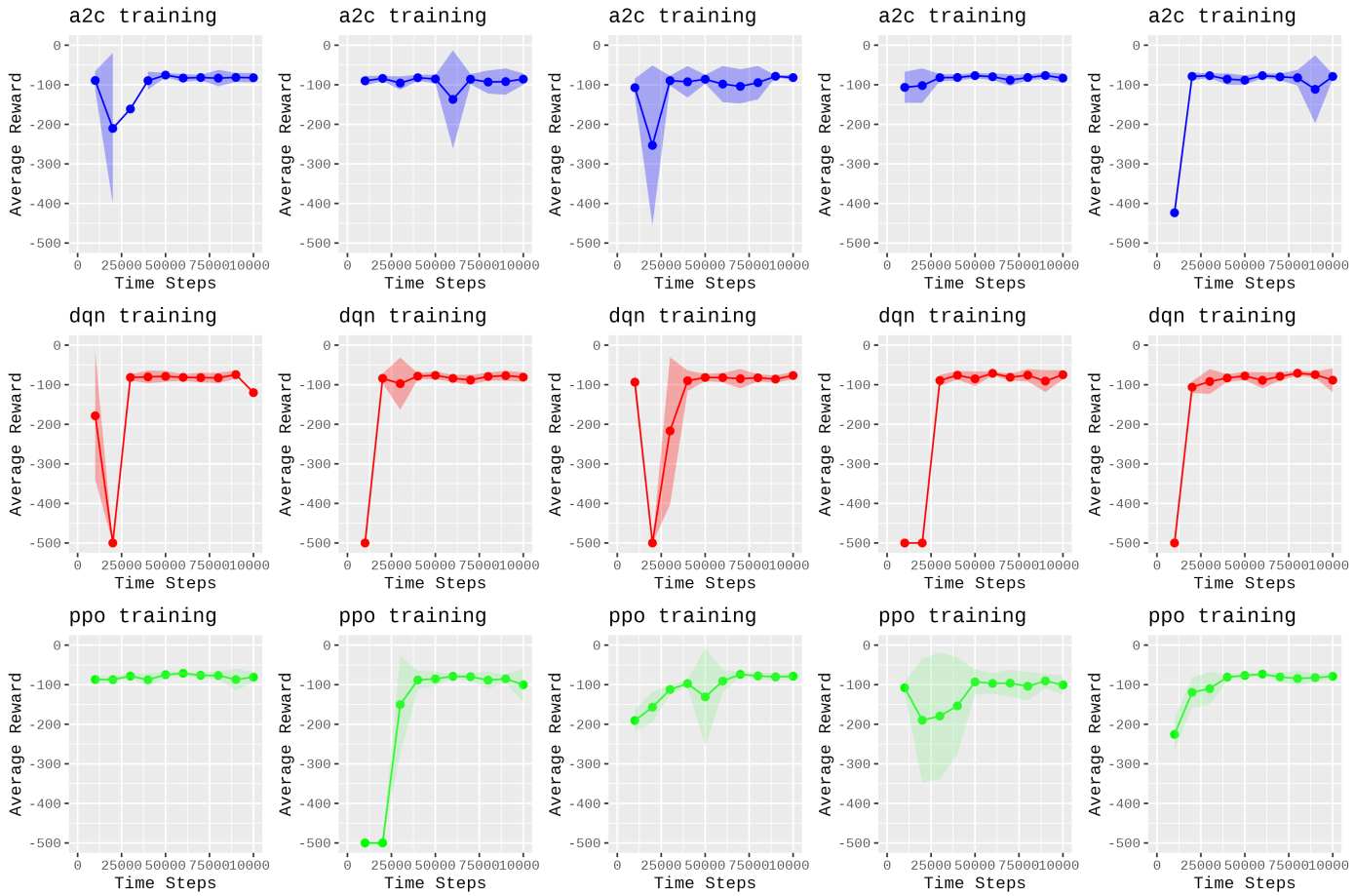

Figure 25: These plots show the individual training runs of the A2C (blue), DQN (red) and PPO (green) algorithms on the **Acrobot** RL task. Each algorithm undergoes five individual training runs, each of which plotted separately. Data points are calculated by taking the mean rewards obtained from the check-pointed policy at the respective timestep, which is evaluated over $E = 10$ evaluation episodes.

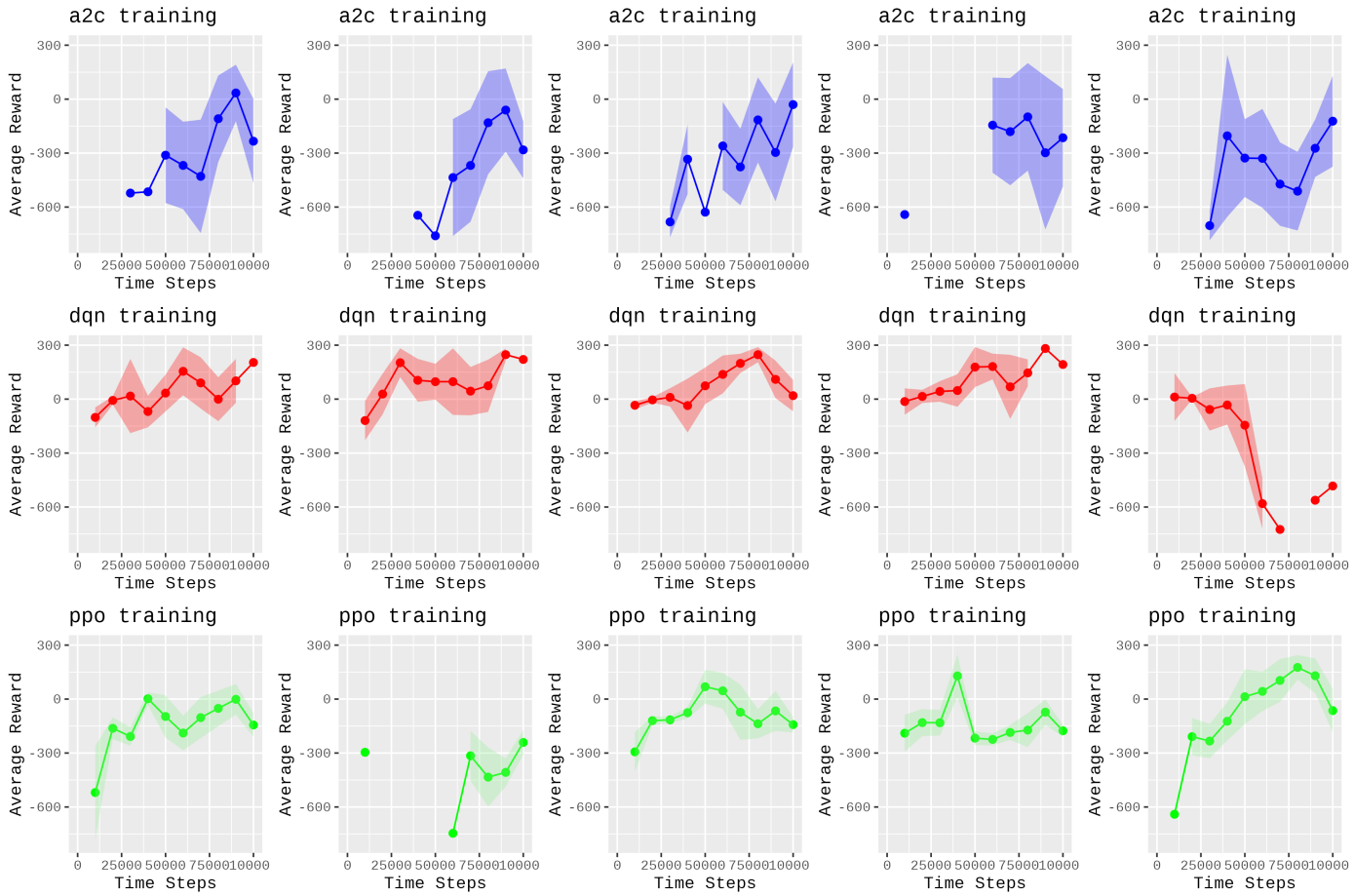

Figure 26: These plots show the individual training runs of the A2C (blue), DQN (red) and PPO (green) algorithms on the **Lunar Lander** RL task. Each algorithm undergoes five individual training runs, each of which plotted separately. Data points are calculated by taking the mean rewards obtained from the check-pointed policy at the respective timestep, which is evaluated over $E = 10$ evaluation episodes.