University of Sheffield

Multi-faceted Performance Metrics for Reinforcement Learning

Reabetswe Michael Nkhumise

Supervisors: Prof Tony Prescott & Dr Aditya Gilra

March 20, 2026

"In the larger scheme of things, greatness lies in acknowledging one's imperfections while continuing to persist." — a personal reflection

Acknowledgments

The Beginnings. My earliest childhood memories involve wastefully mixing my mother's makeup and lotions while pretending to be a scientist, or marvelling at the idea of replacing my uncle's paralysed hand with a robotic one. I have always wanted to be a scientist and an inventor. So, I devised a plan: learn how to create machines and systems, automate them, and eventually make them think for themselves. This led me to pursuing a bachelor's degree in Mechanical Engineering, a master's degree in Robotics, and ultimately a PhD in Reinforcement Learning (RL).

Family. My family has always been supportive — finding my childhood ideas about the future amusing, tolerating my habit of over-explaining things and dismantling electronics to see how they worked, and encouraging me to study internationally to gain exposure to new ideas and perspectives. My mother, Sinah, and my two brothers, Salthiel and Ofentse (posthumous), have been and continue to be a profound blessing in my life. This of course includes my aunt, Evah, and nephew, Katlego.

Friends. I made new friends after moving to Sheffield, and I now consider many of them to be family. These include Vicki Neath, Dave Neath, Anne Kingstone, Dr Lawrence Schobs, Pawel Pukowski, Zeyu Song, Baslin James, and Dr Mohamed S. Talamali. They made my social life during my PhD a truly joyful experience and helped create unforgettable memories that I will always cherish. I am also grateful to my friends back home — Klaas Molapisi, Buyi Mzizi, Lucky Mthombeni, Tshiya Moagi, Tumisang Selamulela, Tebogo Ngoasheng, Maketele Khalo, Howard Williams and Itumeleng Malebye — who have stayed in touch and shared their own life journeys with me.

The Colleagues. I had the privilege of working in both the Machine Learning and Robotics groups, which allowed me to engage with diligent and exceptional colleagues such as Dr Juan Jose Giraldo Gutierrez, Dr Chunchao Ma, Dr Chao Han, Dr Shuo Zhou, Dr Sina Tabakhi, Dr Luca Manneschi, Dr Matt Ellis, Dr Stanley C. Obilikpa, Dr Genki Miyauchi, Dr Bhoomika Gandhi, Dr Subha Veeramani, Dr Lawrence Schobs, Pawel Pukowski, Zeyu Song, Baslin James, Dr Mohamed S. Talamali, Louise Caffrey and Alex Lucas. They shared their knowledge and advice, listened to my concerns, and offered feedback, encouragement, and support. Without their generosity, my journey would undoubtedly have been very different.

The Supervisors. I am thankful to Prof. Eleni Vasilaki for helping me develop a

strong foundation in RL and for giving me the opportunity to be a teaching assistant for her modules. I also appreciate Prof. Tony Prescott for his guidance and for agreeing to become my primary supervisor after I changed research groups. I'm indebted to Prof. Sanja Dogramadzi for generously lending me a computer from her lab when mine abruptly began failing. I'm obliged to Dr James Law for his guidance and always going an extra-mile to create research opportunities for me. Above all, I am especially grateful to Dr. Aditya Gilra for his guidance, patience, and remarkable attention to details. Although he relocated to the Netherlands and could no longer serve as my primary supervisor, he continued to meet with me virtually every week. I could not have asked for a better mentor.

Dearest Wife & Daughter. My wife and daughter have been loving, patient, and supportive throughout this journey. I am deeply thankful to my wife, Koketso Nkogatse, for choosing to relocate with me to Sheffield. We were blessed with an incredible daughter during my studies, and it has been a joyful experience. Thank you for listening to my worries, lifting my spirit, and standing with me through the highs and lows. This journey would have borne no fruit without your love and support. I am truly fortunate to have you both by my side.

Abstract

Deep reinforcement learning (DRL) is a promising approach for endowing robotic agents with full autonomy. Despite its wide successes, DRL lacks a comprehensive theory, and empirically, it suffers from inconsistent reward-based performance (i.e. cumulative rewards) across benchmarks. To address this, we develop a *systematic evaluation methodology* that examines and compares DRL algorithms across multiple performance dimensions instead of relying solely on rewards. The framework analyses algorithms over *exploration, robustness* and *long-term consequences* aspects of algorithm performance for better insight.

An efficient RL agent effectively addresses challenges of exploration, generalisation and long-term consequences. In exploration, we develop *Effort of Sequential Learning* (ESL) and *Optimal Movement Ratio* (OMR) to capture exploration efficiency. Typically, a reward-based metric such as regret is employed for evaluating exploration. As complementary metrics, we introduce ESL — the relative distance travelled by an algorithm in the policy space to discover an optimal policy — and OMR (the fraction of movements in the policy space the algorithm takes to effectively reduce an analogue of regret).

Directly capturing generalisation can be difficult, so we study robustness of RL algorithms and task complexity to capture certain aspects of generalisation. Robustness is required for an algorithm to generalise, while task complexity dictates the task ordering in curriculum learning which significantly impacts generalisability. We examine long-term consequences via cumulative rewards and demonstrations. This is because cumulative rewards do not always qualitatively represent the agent's desired behaviour. Thus, observing the behaviour of the agent while executing the learned policy is imperative for validating its performance.

Finally, we demonstrate the utility of our proposed *evaluation methodology* by employing it in a robotic task to assess and compare RL algorithms suitable for robotic manipulation. Our framework is rich and principled, particularly useful where strictly reward-based performance metrics can misrepresent true task success.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Summary: Reinforcement learning (RL) holds significant promise for developing autonomous agents applicable across diverse domains, particularly robotics. However, RL remains theoretically underdeveloped in some areas and empirically suffers from poor reproducibility. To address these challenges, ***we investigate methods to capture the multi-faceted performance of RL algorithms and propose a systematic framework for their evaluation****. This approach aims to enhance the understanding and interpretability of RL methods, and can be valuable in domains such as robotics, where different aspects of agent behaviour are typically assessed.*

In recent years, reinforcement learning (RL) has attracted plenty of research interest. Ever since an RL algorithm (notably DQN (Mnih et al., 2013)) autonomously played arcade video games and surpassed human performance (Mnih et al., 2015), RL has continued to be a promising framework for developing autonomous agents (Silver et al., 2017; François-Lavet et al., 2018; Ibarz et al., 2021). Autonomous agents are systems that perceive their environments through sensors and act upon them through actuators (Russell & Norvig, 2010). They are important in many applications such as chatbots, self-driving vehicles, robotics, etc. (Dulac-Arnold et al., 2019; Terven, 2025). For example in robotics, autonomous agents can automate and enhance production and operation especially in major industries (Hentout et al., 2023) such as agriculture (Zhang et al., 2023), manufacturing (Thomas et al., 2018; Bingran et al., 2020), mining (Marshall et al., 2016), and healthcare (Yang et al., 2018; Argall, 2018) among others (Thrun, 2002; Siciliano et al., 2008; Chen et al., 2022).

The appeal of RL lies in its learning paradigm: agents gain task knowledge through rewards and penalties received during interaction with the environment. This is done without needing the practitioner to explicitly specify *how the task should be performed* (Kaelbling et al., 1996). Modern RL is often referred to as *deep RL* (François-Lavet et al., 2018), since it leverages the representational and generalisation capabilities of

neural networks (Silver et al., 2018; Tassa et al., 2018). Despite its appeal and successes, deep RL presents theoretical and empirical challenges. It is theoretically incomplete (Osband et al., 2019; Laidlaw et al., 2023) and it suffers from poor *reproducibility* (Henderson et al., 2018). In this thesis, *reproducibility* refers to the ability to obtain statistically similar results when deploying the same RL algorithm under the same conditions (i.e. environment and hyperparameters).

The theoretical incompleteness and poor reproducibility of deep RL methods undermine the *interpretability* of RL algorithms. That is, *the understanding of why RL algorithms behave the way they do* (including *how they make decisions* and *what they learn about the environment*) is hampered. This, consequently, constrains confidence in the practical deployment of RL algorithms (Grothoff et al., 2022). In response, researchers have introduced benchmarks (Bellemare et al., 2013; Tassa et al., 2018; Cobbe et al., 2020) and called for the investigation of novel evaluation metrics (Whiteson et al., 2011; Henderson et al., 2018; Dulac-Arnold et al., 2019; Adkins et al., 2024), with the aim of enhancing the systematic understanding (Conserva & Rauber, 2022) and *interpretability* of comparative analyses of RL algorithms.

Using benchmarks works to some extent; however, it is often unclear which algorithm won across the benchmarks (Henderson et al., 2018), especially when using solely a reward-based performance such as cumulative reward (return). This type of single-faceted performance metric can fail to sufficiently capture the efficiency of RL algorithms (Osband et al., 2019). For this reason, some researchers have suggested introducing multi-faceted performance metrics (Whiteson et al., 2011; Henderson et al., 2018; Adkins et al., 2024), which can assess various aspects of performance to facilitate interpretability. However, this approach raises the following questions: *What can be described as reasonable dimension(s) of agent performance?* and *What is an appropriate empirical methodology to evaluate the algorithms?*

Answers to these questions can help advance the field. For instance, with pertinent performance dimensions known and an appropriate evaluation methodology developed, RL algorithms can be analysed in a more nuanced and systematic manner. This can be effective at revealing trade-offs and dependencies that are obscured by an aggregate metric like cumulative reward (Agarwal et al., 2021; Adkins et al., 2024). It can be useful, particularly in robotics, where the agent behaviour is examined — to avoid damage or unsafe actions (Ibarz et al., 2021; Tang et al., 2025) — in conjunction with task success. A multi-faceted approach that assesses various aspects of agent behaviour can be a great and key fit in RL applications, especially in robotics (Xu et al., 2020). It can ultimately inspire new RL approaches with better decision-making autonomous agents.

1.2 Background

Reinforcement learning (RL) is a framework that attempts to identify an optimal policy (controller or control law) that achieves the goal of a system efficiently (Thrun, 1992). It attains this by maximising an objective function through trial-and-error (Sutton & Barto, 2018). Formally, RL can be understood as a collection of algorithms designed to solve sequential decision-making problems under uncertainty (Givchi, 2021), often modelled as *Markov Decision Processes* (MDPs) (Russell & Norvig, 2010). MDPs are stochastic processes that consist of a set of states $\mathcal{S}$ (state space), a set of actions $\mathcal{A}$ (action space), transition dynamics $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and a discount factor $\gamma \in [0, 1)$ (Sutton & Barto, 2018).

Typically in an MDP, an agent is considered to interact with the environment in discrete timesteps (Lillicrap et al., 2016). At each timestep $t \in \mathbb{N}$, the agent observes a state $s_t \in \mathcal{S}$, executes an action $a_t \in \mathcal{A}$ to transition between states $s_t \to s_{t+1}$, and receives an associated reward $\mathcal{R}(s_t, a_t)$ from the environment, as illustrated in Figure 1.1. The probability of transitioning to the next state $s_{t+1} \in \mathcal{S}$ depends solely on the current state-action pair (s_t, a_t), thus $T(s'|s, a) = \mathbb{P}(s_{t+1} = s' \mid s_t = s, a_t = a)$ for $t \geq 0$. The agent learns to take actions that maximise the cumulative reward, called *return*, while taking into account the uncertainty in state transitions.

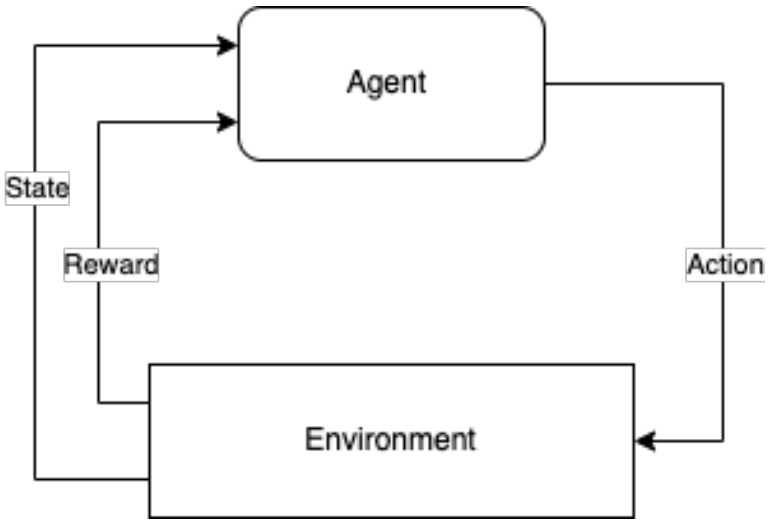


Figure 1.1: *Illustration of agent-environment interaction.*

Success of (deep) RL. RL has had success in playing video games to human proficiency (Silver et al., 2017; Jaderberg et al., 2019), and even surpassing human-level in strategic board games like chess (Silver et al., 2018). Additionally, it has been utilised in tuning databases and computer systems (Wang et al., 2021; Basu et al., 2019), creating recommender systems (Lin et al., 2023), managing energy systems (Yang et al., 2020), and controlling robots (Thomas et al., 2018; Ibarz et al., 2021; Chen et al., 2023a). This success is primarily attributed to the capabilities of neural network-based

function approximators (Silver et al., 2018; Tassa et al., 2018). Integrating RL with deep learning techniques (Goodfellow et al., 2016), referred to as *deep RL*, enabled RL to tackle problems with high dimensional state-action spaces (François-Lavet et al., 2018).

Despite this wide success, RL has shown mixed results in robotics (Tang et al., 2025) such that model-based control or hand-designed controllers remain dominant, especially in manipulation tasks (Zhang et al., 2023). Unlike games, robots lack well-shaped reward functions, making it challenging to effectively capture desired behaviours (Ibarz et al., 2021). Furthermore, RL requires collecting trial-and-error samples (experience) for learning, which in robotics presents many choices in how learning is initialised and how to avert unsafe behaviours (Ibarz et al., 2021).

Reward Function. A reward function generates a reward signal that steers an agent towards desirable actions and, consequently, behaviour (Eschmann, 2021; Devidze, 2024). It provides informative feedback to the agent which facilitates learning. Ideally, a reward function encodes the goal, i.e. desired agent behaviour. However, it is non-trivial to define a reward function that aligns with the desired behaviour (Ecoffet et al., 2021; Aleksandrowicz & Jaworek-Korjakowska, 2023; Ibrahim et al., 2024; Devidze, 2024), particularly if it is not intrinsic to the environment (Pathak et al., 2017; Kirk et al., 2023) like robotic tasks (Mohtasib et al., 2021). Moreover, the difficulty of designing a sound reward function increases as the environment becomes more complex (Dewey, 2014; Wang et al., 2020).

Reward functions can introduce uncertainties in the agent's behaviour that can engender misalignment between learned policies and intended objectives. This can lead to undesirable agent behaviours (Andrychowicz et al., 2017; Ecoffet et al., 2019; du Preez-Wilkinson & Gallagher, 2020). These uncertainties often arise from factors such as (Ibrahim et al., 2024):

1. Sparsity - lack or delay of frequent rewards.
2. Deception - poor local optima in the reward landscape that encourage the agent toward globally suboptimal policies.
3. Unintended consequences - unexpected outcomes from agent-environment interplay.
4. Reward hacking - agent maximising rewards without achieving the intended goal.
5. Reward-objective misalignment - imprecise encoding of the true goal.
6. Imbalanced multi-objective trade-offs.

Exploration. Exploration emerges during the agent's trial-and-error interactions with the environment. Initially, the environment is unknown to the agent and the agent has to identify a (near-)optimal policy (Thrun, 1992). To resolve this dilemma, the agent se-

lects actions at random to observe state-action-reward associations, gradually building to varying degrees an internal representation of state-action-reward relationships (Sutton & Barto, 2018). This constitutes acquiring knowledge about the environment and it is similar to how animals and human beings learn (Mehlhorn et al., 2015). While the agent obtains information about the environment, it progressively transitions from prioritising information acquisition to choosing actions that maximise the reward signal (Ladosz et al., 2022).

The process of choosing actions at random and selecting those maximising rewards is known as exploration-exploitation tradeoff (Sutton & Barto, 2018). An agent with an *efficient exploration scheme* benefits from strategically balancing the tradeoff (Thrun, 1992; Mehlhorn et al., 2015). Additionally, an *efficient exploration scheme* intelligently selects actions, as uniform randomness fails in some environments to discover highly rewarding state-action sequences or (near)-optimal policies (Hofer & Gimbert, 2016). Such environments are characterised by high *visitation complexity* — which is the difficulty of visiting all the states to identify rewarding state-action pairs (Conserva & Rauber, 2022) — and are often referred to as "hard-exploration problems" (Bellemare et al., 2016; Ecoffet et al., 2019). In these environments, uniform random actions struggle to gather meaningful experience (i.e. samples or data comprising of states, actions and rewards) and RL algorithms often perform poorly (Ecoffet et al., 2019).

Model-based RL vs Model-free RL. There are two main categories of RL approaches, namely model-based RL (MBRL) and model-free RL (MFRL) (Huang, 2020). MFRL algorithms estimate the optimal policy without explicitly modelling the dynamics of the environment (Ibarz et al., 2021). In MBRL, the dynamic model of the environment (often called the predictive model) is either learned or provided, then used in planning to determine the optimal policy (Moerland et al., 2023). MFRL algorithms do not attempt to model the environment, but instead respond directly to the environment changes and feedbacks (Huang, 2020). This is favourable for many real-world tasks, where the environment dynamics are highly complex to model (Gu et al., 2016). The disadvantage is that MFRL algorithms are sample inefficient compared to MBRL algorithms (especially with a known or estimated model), and thus require copious amounts of data samples to learn a good policy (Schulman et al., 2015; Chua et al., 2018; Valencia et al., 2023).

With the predictive model, MBRL algorithms can train agents with virtual experience, hence drastically reducing the number of interactions with the environment to collect data samples (Huang, 2020). This makes MBRL algorithms have superior sample efficiency (Deisenroth et al., 2013), and the predictive model efficiently enables generalisation to unforeseen situations (Hafner et al., 2020; Ibarz et al., 2021; Valencia et al., 2023). Nevertheless, in large scale applications MBRL algorithms were generally outperformed by MFRL algorithms (Grześ, 2017), but recently MBRL algorithms have become competitive (Chua et al., 2018; Kaiser et al., 2019; Wang et al., 2019; Hafner et al., 2025). Additionally, MBRL algorithms are appealing to robotics, chiefly in real robots, since they can improve safety and reduce wear due to minimal environment in-

teractions (Valencia et al., 2023). The disadvantage is that imperfections in the model can be exploited by the agent, which can lead to compromised performance (Ibarz et al., 2021).

Efficient RL agent. According to Osband et al. (2019), an *efficient RL agent* addresses the following three challenges concurrently:

1. Generalisation - performing well in novel situations beyond the training data.
2. Exploration - gathering meaningful experience to learn from.
3. Long-term consequences - considering hyperopic effects of actions on the task beyond a single timestep.

We appoint the agent's performance on these challenges as the basis for a meaningful and informative criterion to evaluate its capabilities. However, because *deep RL* lacks a comprehensive theory (Osband et al., 2019; Conserva & Rauber, 2022; Laidlaw et al., 2023) and is still in its early stages (LeCun et al., 2015; Osband et al., 2019), reliably evaluating and comparing RL algorithms in a principled manner using these challenges presents ongoing difficulties. The work reported in this dissertation sets out to address these difficulties by investigating suitable metrics for assessing *generalisation*, *exploration* and *long-term consequences* of RL algorithms. Ultimately, we develop a systematic, multi-faceted evaluation framework that aims to facilitate clearer insights into the strengths and limitations of the algorithms.

1.3 Problem Statement

Deep reinforcement learning (deep RL) presents both theoretical and empirical difficulties despite its wide success. Its current theory is still in its infancy (Osband et al., 2019; Laidlaw et al., 2023), it is sample-inefficient (Ye et al., 2021), it demonstrates limited zero-shot generalisation (Kirk et al., 2023), and it suffers from poor reproducibility during empirical evaluations (Henderson et al., 2018; Patterson et al., 2024). For example, different codebases of the same algorithm, as well as trial groupings (e.g. N vs. two $N/2$ splits), can produce outcomes that differ as if from utterly distinct algorithms (Henderson et al., 2018). Moreover, Islam et al. (2017) and Henderson et al. (2018) showed that performance can vary drastically with random seeds, number of trials, codebases, neural network architectures, hyperparameters, and reward scales.

In response to these challenges, some researchers have introduced benchmarks (Bellemare et al., 2013; Tassa et al., 2018; Osband et al., 2019; Towers et al., 2024) while others have employed them (Duan et al., 2016; Wang et al., 2019; Fujimoto et al., 2019) to evaluate and gain better insight into existing RL algorithms. However, these benchmarks are often criticised for lacking a principled theoretical basis (Oller et al., 2020; Foster et al., 2021; Conserva & Rauber, 2022) for task selection. In most cases, environments are selected using heuristics or domain expertise rather than structural criteria (Jordan et al., 2024; Voelcker et al., 2024). Consequently, benchmark suites

may not provide controlled or well-distributed coverage of task difficulty or types of learning challenges for the algorithms (Duan et al., 2016). For example, *bsuite* (Osband et al., 2019) offers tasks intended to probe specific aspects of agent behaviour, nevertheless there is no theoretical analysis demonstrating how reliably these environments isolate or quantify the intended properties. Similarly, *OpenAI Gym* (Brockman et al., 2016) provides a wide range of tasks, but lacks a structural framework establishing which algorithmic capabilities are being systematically assessed.

This means that the performance differences across such benchmarks can be difficult to interpret in terms of specific algorithmic strengths, thereby limiting insight into the mechanisms governing algorithmic behaviour. Rather than relying solely on the benchmarks, other researchers advocate for investigating better metrics for assessing RL algorithms (Whiteson et al., 2011; Henderson et al., 2018; Sünderhauf et al., 2018; Chan et al., 2020; Ahmed et al., 2020; Adkins et al., 2024). Such metrics can offer a multi-faceted analysis of algorithms to help deepen our understanding (Adkins et al., 2024). With this overview established, we now focus on the specific challenges of long-term consequences, exploration, and generalisation in RL.

Long-term consequences. According to Sutton & Barto (2018), a single timestep consequence of an action is *a reward and a new state* that the agent finds itself in. This implies that the long-term consequences of an action comprise of the entire sequence of future states and rewards it engenders. With this, we can understand *long-term consequences as the behavioural evolution (state sequence) and evaluative feedback (reward sequence) ensuing from an action.* For this reason, long-term consequences can be understood interchangeably as long-term behaviour (or behavioural evolution) of the agent, and a standard metric for evaluating them is the expected cumulative reward, i.e. *expected return* or *objective function* (Aleksandrowicz & Jaworek-Korjakowska, 2023).

The objective function is expressed distinctly depending on whether the MPD has a finite or infinite (planning) horizon. In a finite horizon MDP where there is a fixed window of timesteps ($0 \leq t \leq T$), the objective function is $J_\mu^\pi = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \mathcal{R}(s_t, a_t)\right]$. $\mathbb{E}_\pi[\cdot]$ denotes the expectation over the probability distribution of trajectories induced by the policy π, while μ is the initial state distribution. In an infinite horizon MDP, where $0 \leq t \leq \infty$, future rewards are discounted with discount factor $0 < \gamma \leq 1$, and the objective function is $J_\mu^\pi = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)\right]$.

Ideally, using the *expected return* as a long-term consequences (behaviour) metric is sufficient, and most research emphasis has been on the best statistical ways to report it (Henderson et al., 2018; Colas et al., 2019; Chan et al., 2020; Agarwal et al., 2021). For example, using a small number of trials ($N < 5$) has been criticised to be inadequate and possibly misleading when reporting performance (Henderson et al., 2018). Nevertheless, beyond statistical considerations, it still remains largely unexplored how uncertainties in the reward function impact whether the expected return is a reliable measure of performance (Amodei et al., 2016). Expected returns do not

always qualitatively represent the agent's desired behaviour (Henderson et al., 2018; du Preez-Wilkinson & Gallagher, 2020). This arises due to the expected return being a numerical summary of the agent's long-term behaviour, without indicating how or why the agent selected its actions — a phenomenon known as *value ambiguity* (Yau, 2024). The discrepancy is particularly pronounced in robotics where high returns may output poor motion quality, like non-smooth or jittering motion (Mahmood et al., 2018; Yuan et al., 2023). This highlights that reward-based performance metrics (i.e. expected return, reward curves), by themselves, can be insufficient as a basis for algorithm evaluation and comparison — primarily in tasks where rewards are not intrinsic to the environment (Andrychowicz et al., 2017; Ecoffet et al., 2019).

Exploration. Exploration is another dimension that heavily influences the overall performance of the agent (Ladosz et al., 2022). It, together with learning strategies, determines the quality of experiences from which the agent learns (Kaelbling et al., 1996; Zhang et al., 2019). Several recent works have advanced exploration techniques (Bellemare et al., 2016; Burda et al., 2019; Eysenbach et al., 2019) and learning methods (Lazaridis et al., 2020; Müller et al., 2021; Li, 2023). However, there remains a lack of consensus over approaches that can quantitatively compare these exploratory processes across RL algorithms and tasks (Seijen et al., 2020; Amin et al., 2021; Ladosz et al., 2022). This is attributed to some methods being algorithm-specific (Tang et al., 2017), while others provide theoretical guarantees for very specific settings (Lattimore & Szepesvári, 2020; Agarwal et al., 2022).

Regret, the performance gap between the agent's policy and the optimal policy (Dick, 2015; Bourel et al., 2020), is often used to assess exploration. However, it is explicitly reward-dependent (Jaksch et al., 2010), and thus captures performance discrepancies with no insight into the agent's behaviour. For example, two agents with distinct policies can have the same regret. While this may be inconsequential in settings without safety and physical constraints (like games), in robotics both performance and the behaviour that lead to it are critical (Kober et al., 2013). Accordingly, a metric that captures the agent's exploratory behaviour is vital, and can complement regret to provide a more comprehensive evaluation of exploration beyond reward-based performance.

Generalisation (Task Complexity and Robustness). In RL, generalisation is a broad concept and its definition is dependent on the scope (Kirk et al., 2023). On a policy-level, it can refer the notion of a learned policy performing well in previously unseen states (Sutton & Barto, 2018) or over planning horizons (Myers et al., 2025) that were not encountered during training — two distinct aspects of generalisation within the same MDP. Additionally, it can describe deploying a trained agent to new MDPs that are outside its training set (Cobbe et al., 2019). On an algorithm-level, generalisation can pertain to the algorithm's good performance across tasks without requiring reconfiguration for each task (Ibarz et al., 2021). These instances fall under *unseen test scenarios*, which draws similarities between generalisation in RL and Supervised Learning (Tran et al., 2022; Paschali et al., 2018).

Typically, generalisation is captured by the gap in the policy performance on samples that diverge slightly from training samples (Xu & Mannor, 2012). This is formally defined as the *generalisation gap* (Kirk et al., 2023):

$$\text{GenGap}(\pi) = J^{\pi}_{\text{train}} - J^{\pi}_{\text{test}} \tag{1.1}$$

where J^{π}_{train} is the return of the policy in the training environment, e.g. RL benchmarks. J^{π}_{test} is the return of the policy in the test environment which shares similarities with the training environment. The test environment can be a real-world environment or modification of the training environment (Kirk et al., 2023; Aleksandrowicz & Jaworek-Korjakowska, 2023).

Generalisation and Task Complexity. The agent's ability to generalise is crucial for success in environments with high variability or stochasticity (Cobbe et al., 2020). Such environments are intuitively known to be difficult to solve, and are common in the real-world (Dulac-Arnold et al., 2019). As a consequence, this has inspired research in examining generalisation in the context of *task hardness* (Justesen et al., 2018; Nichol et al., 2018; Cobbe et al., 2019, 2020; Kirk et al., 2023). *Task hardness (complexity or difficulty)* is the degree of difficulty in solving the task (Furuta et al., 2021). It informs generalisation by enabling principled curricula across varying difficulty levels (Justesen et al., 2018; Cobbe et al., 2019). Such curricula are vital for training agents on tasks that may be too hard to learn tabula rasa — a method known as *curriculum learning* (Narvekar et al., 2020). Agents trained using curriculum learning (Bengio et al., 2009; Narvekar et al., 2020; Narvekar, 2021) typically outperform those trained directly on the most difficult tasks (Bengio et al., 2009; Justesen et al., 2018; Narvekar, 2021).

Nevertheless, without a task hardness theory in deep RL (Conserva & Rauber, 2022), a standardised task complexity metric has not yet been established (Furuta et al., 2021). The lack of such a metric may limit the understanding of generalisation (Narvekar et al., 2020). For example, effective sequencing or ordering of tasks is primary in curriculum learning as it facilitates incremental learning and gradual skill acquisition (Justesen et al., 2018; Cobbe et al., 2019; Narvekar, 2021). It however relies on moderate difficulty gaps between tasks, since large jumps can possibly disrupt the incremental learning and gradual skill acquisition (Narvekar et al., 2020).

Typically, heuristics — such as state-action dimensionality (MacAlpine & Stone, 2018), reward structure (Wu & Tian, 2017) and sample complexity (Narvekar, 2021) — are employed to order tasks (Narvekar et al., 2020). However, they cannot guarantee consistency in maintaining appropriate difficulty gaps between tasks, since they do not fully capture how hard the task is (Conserva & Rauber, 2022; Rajan et al., 2023). This suggests that the comprehension of RL task complexity can positively impact the quality of generalisation (especially attained via curriculum learning), since the task ordering depends on it. Several works on task complexity in deep RL have been published (Oller et al., 2020; Furuta et al., 2021; Conserva & Rauber, 2022; Rajan et al.,

2023), yet the proposed frameworks have not seen ubiquitous adoption across the field. We evaluate some of these methods in robotic tasks to understand task difficulty in RL, as a prerequisite for generalisation.

Generalisation and Robustness. Robustness is related to the algorithm's stability and the likelihood of failure of the learning model (i.e. its reliability) (Paschali et al., 2018). A stable algorithm consistently performs well in the presence of slight changes in the environment (Xu & Mannor, 2012), while a reliable algorithm has repeatable or reproducible performance (Chan et al., 2020). Therefore, a robust algorithm has consistently good and reliable performance across environments (Xu & Mannor, 2012). Algorithmic robustness is considered to be an independent epistemic notion (Freiesleben & Grote, 2023), that is deeply connected with generalisation.

For instance, stable algorithms tend to generalise (Shalev-Shwartz et al., 2009). This is supported by Kawaguchi et al. (2022)'s statement that "robust algorithms generalise well to unseen data for various models." Such a perspective is shared in RL, where an algorithm is often considered robust if it generalises with minimal performance drop to a set of tasks without requiring additional tuning of hyperparameters (Higgins et al., 2017b). Furthermore, studies such as (Xu & Mannor, 2012; Rajeswaran et al., 2017; Song et al., 2019) examined the relationship between generalisation and robustness, with Xu & Mannor (2012) concluding that *robustness is both necessary and sufficient for a learning algorithm to generalise*. These demonstrate that robustness is one of the important conceptual tools to reason about generalisation (Shalev-Shwartz et al., 2009; Zhang et al., 2021b), and it can reveal whether an algorithm has the potential to generalise (Xu & Mannor, 2012).

Studying generalisation in RL can be challenging because the benchmark tasks are often dissimilar to the real-world tasks (Duan et al., 2016; Panaganti et al., 2022) we often apply the algorithms to (Kirk et al., 2023). Furthermore, RL algorithms are quite brittle and can have widely varying performance depending on (1) how their hyperparameters are selected and (2) specific properties of the task (Henderson et al., 2018; Haarnoja et al., 2018; Kirk et al., 2023). The generalisation gap (Equation 1.1) often fails to capture good generalisation, since for instance (Kirk et al., 2023):

1. A random policy is likely to get a generalisation gap of zero, and this does not imply a good performance across environments.

2. If the reward functions are incomparable across environments, then the magnitude of the generalisation gap may not be informative.

For these reasons, we take inspiration from Kuzborskij & Lampert (2018), by examining the robustness of algorithms as a basis for understanding an aspect of the algorithms' generalisability. This aspect can help RL practitioners determine *whether generalisation can potentially occur or not*. Note that there is no evidence or study that suggests that robustness captures generalisation, but robustness is required for generalisation to occur (Xu & Mannor, 2012). In place of studying generalisation directly, we focus

our attention to its critical prerequisites, viz. *Task Complexity* and *Robustness*.

Concluding remarks. In this section, we explored the key details of the challenges an *efficient RL agent* needs to address, namely:- *long-term consequences (behaviour)*, *exploration*, and *generalisation*. We aim to establish how an agent performs across them as the foundation for an informative criterion to assess its capabilities, and thereby, develop a multi-faceted evaluation framework. In the next section, we briefly examine one of the widely studied applications of RL (i.e. robotics) and highlight the key questions that cannot be adequately resolved if a single-faceted performance metric such as cumulative reward (return) is utilised.

1.4 Robotics

Robotics is a valuable field in human development. It is ubiquitous in major industries (Hentout et al., 2023) including agriculture (Zhang et al., 2023), manufacturing (Thomas et al., 2018; Bingran et al., 2020), and healthcare (Yang et al., 2018; Argall, 2018) amongst others (Thrun, 2002; Siciliano et al., 2008; Chen et al., 2022). It can be described as the science of perceiving and manipulating the physical world through computer controlled mechanical devices (Thrun et al., 2005), commonly referred to as *robots*, *robotic systems* or *robotic agents*. Robots can improve productivity, reduce human costs, enhance accuracy, and elevate safety (Han et al., 2023). Furthermore, robotic systems can operate in high-risk environments where human safety is at risk (Liu et al., 2021; Han et al., 2023).

One of the most widely employed robotic systems in industrial fields is a *robotic manipulator* or *robot arm* (Siciliano et al., 2008; Lv et al., 2024). Robotic manipulators specialise in a subset of tasks, where a robotic agent moves things other than itself, called *manipulation tasks* (Mason, 2018). These tasks include but are not limited to component assembly, welding, painting, and packaging (Mason, 2018; Han et al., 2023). Robots have been utilised for manipulation tasks since the 1960s (Billard & Kragic, 2019), and achieving high performance in these tasks is central to fulfilling the promise of robotics (Kroemer et al., 2021). The promise is to automate complex and real-world operations (Siciliano et al., 2008; Koditschek, 2021).

Nevertheless, many robotic tasks are still dependent on human intervention (Chen et al., 2023b), where human beings are burdened with continual support of the tasks. Not only does this introduce undesirable factors such as human error and downtime (Saha, 2014), it also limits the application opportunities of robots. Thus, there is a growing need to deploy autonomous robots that require little or no human involvement (Shahid et al., 2022). However, uncertainties and complex dynamics remain key obstacles to *full autonomy* in robotic agents (Thrun, 2002). Increasing the degrees of freedom (DoF) of a robot complicates its dynamics, making prediction and control more challenging (Nakanishi et al., 2008; Siciliano et al., 2009). Uncertainties arise from many sources including, but not limited to, environment stochasticity, noise in the robots' sensors, and modelling errors of the robotic systems (Chen et al., 2023b).

Dealing with uncertainties and complex dynamics, to achieve desired robot behaviours in dynamic and flexible environments, requires intricate control algorithms (Mathew, 2014; Fabisch et al., 2019). Such control algorithms have to (1) incorporate feedback to be more robust to inaccuracies and noise (Rathbone, 2000), and (2) make sequential decisions that adapt to time-dependent task conditions (Shahid et al., 2022) often present in robotic tasks. RL is a promising paradigm for meeting these demands (Kober et al., 2013; Polydoros & Nalpantidis, 2017; Ibarz et al., 2021; Lobbezoo et al., 2021; Kaufmann et al., 2023; Tang et al., 2025). This stems from the ability of RL algorithms to acquire knowledge about the system and its environment via interactions (Sutton & Barto, 2018), and generalise to produce desired behaviour in new, unseen scenarios (Elguea-Aguinaco et al., 2023). Moreover, RL does not involve deriving explicit control laws and/or system dynamic models, in contrast to classical control approaches (Spielberg et al., 2017).

1.4.1 Empirical evaluation

Although RL is appealing to apply in robotics, whenever results of only a reward-based performance metric (e.g. returns) are presented about the algorithm, the following questions can arise:

1. Was the motion of the robot smooth under the algorithm?
2. Is the algorithm sensitive to system modelling errors or slight environment changes?
3. What is the likelihood of the algorithm failing even after extensive training due to using a different codebase?
4. Are the hyperparameter settings also suitable for a different but related task?
5. What is the likelihood of the algorithm finding (near-)optimal policies?
6. Was the exploration strategy of the algorithm truly effective or incidentally discovered the (near-)optimal policy?

A single-faceted performance metric such as cumulative reward is insufficient to address many of these questions. This underscores the need for an empirical methodology that can adequately address the questions (or majority of them). Such a methodology would be *useful* in practice, and can enable nuanced comparisons between algorithms. We see a multi-faceted performance metric as a suitable candidate for this.

1.5 Objectives and Research Question

The aims of this thesis work are to study methods that can capture the multi-faceted performance of RL algorithms, and develop a systematic evaluation methodology based on them. These will be achieved by adhering to the following steps:

1. Explore state-of-the-art (SoTA) RL algorithms.

2. Investigate evaluation methods that assess exploration, robustness, and long-term consequences of RL algorithms.

3. Consolidate the evaluation methods into a multi-faceted performance metric.

4. Demonstrate the utility of the metric by applying it to a robotic task.

In summary, we want to answer the following **research question**:

> Can a rigorous evaluation methodology that captures *exploration, robustness*, and *long-term behaviour (consequences)* of RL algorithms enhance the understanding and interpretability of their comparative analyses?

1.6 Scope and Limitations

As highlighted in Section 1.3, generalisation, exploration and long-term consequences (behaviour) in RL are intricate constructs, each accompanied by subtleties that complicate how they are characterised and measured. In this thesis, we examine aspects of generalisation via task complexity and robustness. For task complexity, we evaluate current approaches in literature using simple robotic tasks. Robustness is studied via the analyses of reliability and hyperparameter sensitivity

Exploration is assessed from a viewpoint of being a combination of the exploration-exploitation interplay and learning strategy. This is because they jointly shape the experience from which the agent learns (Kaelbling et al., 1996; Zhang et al., 2019), and thus the identification of effective policies (Sutton & Barto, 2018). A reward-based performance metric (i.e. expected returns and reward curves) along with behavioural descriptions of the agents, are considered when studying long-term consequences of RL algorithms. Finally, all the task environments assessed in this dissertation are simulation-based with finite horizons.

1.7 Research Contributions

1. This dissertation proposes a framework for comprehensively analysing RL algorithms across multiple dimensions of performance, namely: robustness, exploration, and long-term consequences. This multi-faceted analysis offers another approach to enhancing current insights into RL algorithms, and it can complement existing benchmarks.

2. The thesis work demonstrates that the current quantitative metric of task complexity in deep RL fails to capture the correct hardness of robotic tasks. There is a mismatch between the task hardness rankings offered by a SoTA metric and expectations grounded in classical control theory.

3. Metrics for capturing exploration efficiency are introduced. These are supported

with theoretical justification and validated empirically. The metrics are also extended to high-dimensional continuous settings.

4. An in-depth comparative analysis of SoTA RL algorithms in a robotic task is presented using the proposed evaluation framework. From this, we learn that the algorithm SAC (Haarnoja et al., 2018) is more efficient for our robotic task setting in comparison with DDPG (Lillicrap et al., 2016) and TD3 (Fujimoto et al., 2018).

1.8 Dissertation Outline

The dissertation is structured as follows: Chapter 2 encompasses a review of SoTA RL algorithms. It describes key concepts that are prevalent in deep RL methods. Chapter 3 is a reproduction of the published article (Nkhumise et al., 2025) with slight modifications. It proposes exploration efficiency metrics. Chapter 4 introduces simple robotic tasks that are used for task complexity analysis. This chapter is an excerpt from our conference paper (Nkhumise et al., 2026). In Chapter 5, we study methods for reliability and hyperparameters sensitivity analysis. Chapter 6 offers a systematic demonstration of our proposed multi-dimensional evaluation framework in a high-dimensional continuous robotic environment. Chapter 7 offers a discussion of the entire thesis work, including a summary and recommendations for future work.

Chapter 2

RL State of the Art

2.1 Introduction

Problem settings in RL can be categorised into two main classes (Sutton & Barto, 2018), namely: *tabular* and *non-tabular* settings. Tabular settings are characterised by finite, discrete state and action spaces with low dimensionality, where the policy can be represented by a lookup table (Sutton & Barto, 2018). In non-tabular settings, the number of states and/or actions is very large and potentially infinite (Conserva & Rauber, 2022). This setting so far relies on the use of function approximators to represent the policy (Tassa et al., 2018; Furuta et al., 2021; Morales et al., 2021). Continuous control problems, where actions are continuous and often high-dimensional (Duan et al., 2016), encompass robotic tasks (Fabisch et al., 2019) and fall under the non-tabular RL setting. Thus, it is essential to focus this thesis work on approaches suitable for non-tabular RL settings.

This chapter explores RL concepts and algorithms pertinent to non-tabular settings. We begin with a description of basic elements of MDPs in Section 2.2, followed by an account about neural networks and their applications in RL in Section 2.3. These are an integral part of modern RL methods. A simple non-exhaustive RL taxonomy is provided in Section 2.4, and a discussion of some RL algorithms used in the thesis are described in Section 2.5.

2.2 RL Formalism

As previously mentioned in Section 1.2, an MDP is a stochastic process that models sequential decision-making problems, formally defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T})$ (Kaelbling et al., 1998). Some formulations include the discount factor $\gamma \in [0, 1)$, giving $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$ (Rosenfeld et al., 2018; Kroemer et al., 2021), while others include the initial state distribution $\mu \in \mathcal{P}(\mathcal{S})$, resulting in $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \mu)$ (Silver et al., 2014). As defined in Section 1.2, $\mathcal{S}$ is state space, $\mathcal{A}$ is action space, $\mathcal{R}$ is reward function, and $\mathcal{T}$ is transition dynamics. A *policy* is a function that maps states to actions in the MDP,

i.e. $\pi : \mathcal{S} \to \mathcal{A}$. Each policy has an associated expected return (cumulative reward) when evaluated from a given state (called *state value*) or state-action pair (called *state-action value*) (Sutton & Barto, 2018). A *value function* maps states, or state-action pairs, to state values, or state-action values, in the MDP given that the agent uses the policy — i.e. $V_\pi : \mathcal{S} \to \mathbb{R}$ or $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The goal of the MDP is to find a policy that maximises the expected return (Bellman, 1957), which coincides with learning value functions that endow maximal values to states or state–action pairs (Puterman, 1990).

2.2.1 Policy

In general the policy can either be time-dependent, denoted as π_t, where it is referred to as a non-stationary policy. Or it can be time-invariant, denoted simply as π, where is it referred to as a stationary policy. During task execution, the agent selects actions in response to observed states, resulting in a sequence of states and actions $h_t = (s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}, s_t)$, called a trajectory or rollout. The policy can select an action a_t based on its trajectory history h_t (Laroche et al., 2022), in which case it is called a non-Markov policy; or it can select an action a_t based solely on the current state s_t, in which case it is called a Markov policy.

In every MDP there exists at least a single optimal Markov policy (Sutton & Barto, 2018). The studies reported in this thesis employ *stationary Markov policies* $\pi(a_t|s_t)$, since for any policy there exists a stationary Markov policy with an equivalent value function (Syed et al., 2008). Note that often in literature, a stationary Markov policy is simply referred to as just a stationary policy, implicitly assuming that it is also a Markov policy. Stationary Markov policies can be deterministic, where the policy issues the same action for a given state, or stochastic — with a distribution over the action space from which actions are sampled for a given state.

2.2.2 Value Functions

The value function that maps states to state values is called the state value function $V_\pi(s)$, while the state-action value function (or Q-value function) $Q_\pi(s, a)$ maps state-action pairs to state-action values. As highlighted in Section 1.3, MDPs can either be finite-horizon (where the trajectory is finite, i.e. $h_{t:T} = (s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}, s_T)$) or infinite-horizon (i.e. $h_{t:\infty}$), thus $V_\pi(s)$ can be defined in each horizon as (Sutton & Barto, 2018):

$$V_\pi(s) = \begin{cases} \mathbb{E}_{a\sim\pi, s\sim\mathcal{T}} \left[\sum_{t=0}^{T} \mathcal{R}(s_t, a_t) \mid s_0 = s \right], & \text{for finite MDP} \\ \mathbb{E}_{a\sim\pi, s\sim\mathcal{T}} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \mid s_0 = s \right], & \text{for infinite MDP} \end{cases} \tag{2.1}$$

while $Q_\pi(s, a)$ is

$$Q_\pi(s,a) = \begin{cases} \mathbb{E}_{a\sim\pi, s\sim\mathcal{T}}\left[\sum_{t=0}^{T} \mathcal{R}(s_t,a_t) \mid s_0 = s, a_0 = a\right], & \text{for finite MDP} \\ \mathbb{E}_{a\sim\pi, s\sim\mathcal{T}}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t,a_t) \mid s_0 = s, a_0 = a\right], & \text{for infinite MDP} \end{cases} \tag{2.2}$$

Equations 2.1 and 2.2 are known as the *Bellman Equations* (Howard, 1960), and are fundamental equations for solving RL tasks (Wang et al., 2024). Note that $V_\pi(s) = \mathbb{E}_{a\sim\pi}[Q_\pi(s,a)]$ and it is related to the *expected return* as follows (Kakade & Langford, 2002):

$$J_\mu^\pi = \mathbb{E}_{s\sim\mu}[V_\pi(s)] \tag{2.3}$$

where $\mu(s)$ is the initial state distribution. An optimal policy π^*, which generates the highest expected return, has a corresponding $V^*(s) = \max_\pi V_\pi(s)$ and $Q^*(s,a) = \max_\pi Q_\pi(s,a)$.

2.2.3 Reward Signal Settings

According to the *reward hypothesis* (Bowling et al., 2023; Abel et al., 2024), all goals can be expressed by the maximisation of the reward signal. Thus, the reward signal is an underlying component of RL that every algorithm must address. There are two types of reward signal settings, namely: *dense* and *sparse* rewards settings (Pathak et al., 2017; Mohtasib et al., 2021). Sparse-rewards settings are characterised by the existence of few states in the state space which issue a positive reward signal (Daaboul, 2020), while the rest issue a non-positive reward signal. Typically the agent only receives positive feedback when the goal state has been successfully achieved. For this reason, sparse rewards are often considered to be *unshaped* reward signals (Andrychowicz et al., 2017). In sparse-rewards settings, it is unlikely the agent stumbles into the goal state through random exploration (Pathak et al., 2017). This makes learning difficult, however the benefit is that it describes the success scenario accurately and concisely (Andrychowicz et al., 2017).

In dense-rewards settings, the environment is augmented with additional reward features shaped (or designed) to aid it with issuing continuous meaningful feedback in all the states (Daaboul, 2020). This type of reward signal was introduced with the purpose of simplifying learning (Mataric, 1994), since the agent continuously receives feedback about its proximity to the goal state. Albeit dense reward signals accelerate learning, they are prone to misalignment with the intended task objectives (Andrychowicz et al., 2017; Ibrahim et al., 2024), as formerly discussed in Section 1.2. This can lead the agent to maximising the expected return without truly accomplishing the task. RL tasks are commonly characterised by either the dense or sparse reward signals, depending on the context.

2.3 Neural Networks (Function Approximators)

Neural networks (NN) are universal function approximators (Goodfellow et al., 2016) at the heart of success of modern RL algorithms (Mnih et al., 2015; Tassa et al., 2018; Furuta et al., 2021). They are typically utilised to approximate the policy and/or value functions (Sutton & Barto, 2018). They can be described as computational models comprising of interconnected processing units, called nodes or neurons, that together loosely emulate the structure and function of biological neural systems (Gurney, 1997; Wu & Feng, 2018). Their goal is to approximate a target function $f : \mathcal{X} \to \mathcal{Y}$ by learning the parameter values $\theta \in \mathbb{R}^d$ such that the resulting function $\hat{f}_\theta$ closely approximates f (Goodfellow et al., 2016). Compared to other function approximators, NN can flexibly approximate complex nonlinear functions, including non-smooth mappings, and can be efficiently scaled to handle high-dimensional input spaces — where $\mathcal{X} \subseteq \mathbb{R}^n, \quad n \gg 1$ (Chua et al., 2018; Moerland et al., 2023).

NN consist of interconnected nodes that are organised into layers to form an architecture portrayed in Figure 2.1. Each node denotes a specific output function called an activation function. The activation function introduces nonlinearity, which enables the NN to capture complex relationships within the data (Bishop, 2006; François-Lavet et al., 2018; Kunc & Kléma, 2024). The connection between any pair of nodes is characterised by a weight that controls the strength of the signal passing through the connection (Wu & Feng, 2018). The layers of NN transform inputs into representations of varying levels of abstraction (Erhan et al., 2009). The first and last layers are referred to as input and output layers, respectively, while the other immediate layers are collectively called hidden layers (Goodfellow et al., 2016).

To optimise parameters θ, NN generally use gradient descent via a learning algorithm called *backpropagation* (Rumelhart et al., 1986; Widrow & Lehr, 2002). Backpropagation determines gradients of the empirical loss or error $\mathcal{L}$ with respect to the network parameters. $\mathcal{L}$ is the discrepancy between the network's output $\hat{f}_\theta(x)$ (where $x \in \mathcal{X}$) and the target function output $y \in \mathcal{Y}$. Typically, this is expressed as,

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L} \tag{2.4}$$

where η is called the learning rate. During training, the error $\mathcal{L}$ is propagated backward from the output layer towards the input layer. To improve the expressivity of NN, i.e. their ability to approximate a wide range of functions, often more hidden layers and more neurons within these layers are added (Goodfellow et al., 2016). Note that adding more hidden layers is referred to as increasing the *depth* of the model — a notion from which the term *deep learning* originates (Goodfellow et al., 2016).

There are several types of NN architectures, including feedforward and recurrent networks, among others. Feedforward networks form directed acyclic graphs without feedback connections, i.e. outputs of any layer cannot be fed back into preceding layers. In

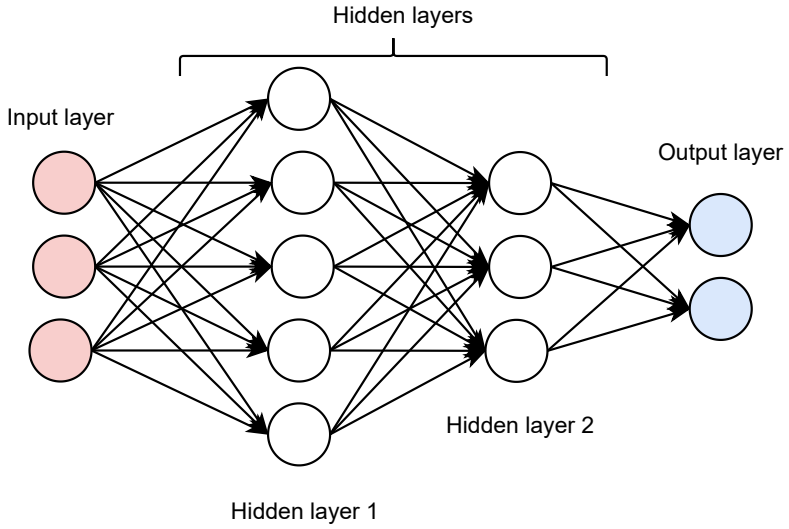


Figure 2.1: *Illustration of the structure of a neural network (NN) model. The model consists of nodes, organised in layers, that are interconnected. Information or signal travels from the input layer to the output layer. The number of hidden layers and number of nodes within these layers can take arbitrary values depending on the task.*

contrast, recurrent networks introduce cyclic feedback connections to model sequential or time-dependent data (Goodfellow et al., 2016; Salehinejad et al., 2017). Convolutional neural networks are a subclass of feedforward network that are particularly well suited for image or visual data (François-Lavet et al., 2018). In this thesis, we mainly employ RL algorithms that use standard feedforward networks (e.g. multilayer perceptrons (MLPs)), as the tasks considered do not require modelling long-term temporal dependencies or processing visual input data.

Although NN models can approximate complex nonlinear functions such as policy and/or value functions in RL, they induce some noise on their outputs (i.e. approximation noise) (Thrun & Schwartz, 2014; Fujimoto et al., 2018; DeVore et al., 2021). This approximation noise can lead to overestimation of output values that can diminish the performance of RL agents (Thrun & Schwartz, 2014). For example, in value functions, the estimation error can randomly engender bad states or state-action pairs to be estimated as high value, resulting in poor policies (Fujimoto et al., 2018). We will discuss later in this chapter how some RL algorithms address this issue.

2.3.1 Autoencoders and Variational Inference

Other useful areas of application of NN include autoencoders — for data compression and feature extraction (Michelucci, 2022; Bank et al., 2023) — and variational inference, for probability density estimation (Doersch, 2016; Bank et al., 2023). Often settings in RL, especially real-world settings, have high-dimensional state-action spaces that are computationally expensive to work with. In such settings, autoencoders can be utilised

for dimensionality reduction (Lange & Riedmiller, 2010). Additionally, in such settings, variational inference can be employed, especially by model-based RL (MBRL) methods, to estimate the dynamic model of the environment (Ha & Schmidhuber, 2018). Below we provide a succinct description of autoencoders and variational inference, since we deploy them in the later chapters.

Autoencoder. An autoenconder is a neural network model trained to reproduce its input data at the output layer (Goodfellow et al., 2016). It was introduced by Rumelhart et al. (1986) with the aim of learning to reconstruct the input with low error, thus capturing *informative* representation of the input. It achieves this by taking an input vector $\mathbf{x} \in \mathbb{R}^d$, map it to a hidden (latent) representation $\mathbf{z} \in \mathbb{R}^{d'}$, and then map back $\mathbf{z}$ to a reconstructed vector $\tilde{\mathbf{x}} \in \mathbb{R}^d$ in the input space (Vincent et al., 2008) — as illustrated in Figure 2.2. In doing this, the model extracts important features of the data by learning its underlying structure, and represents the features compactly within the latent representations (encodings) $\mathbf{z}$ (Michelucci, 2022). This process is key in dimensionality reduction as it converts the input vector $\mathbf{x}$ into $\mathbf{z}$, where the vector dimensions are reduced from d to d' (with $d > d'$).

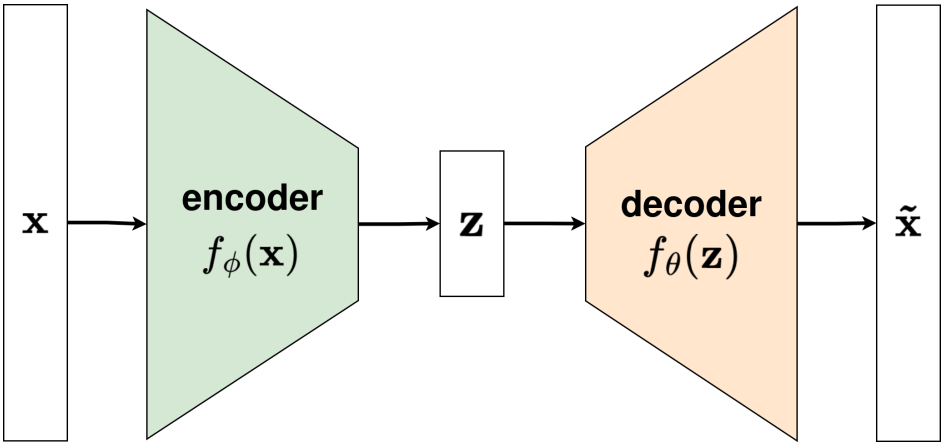


Figure 2.2: *General structure of an autoencoder. The encoder that takes* $\mathbf{x} \in \mathbb{R}^d$ *and the decoder that produces* $\tilde{\mathbf{x}} \in \mathbb{R}^d$ *are neural networks with mirrored architectures, where the order of the neural network layers in the decoder are in reverse to those in the encoder.*

For example, an autoenconder can be applied to represent high-dimensional image input into low-dimensional encodings. This simplifies learning and improves convergence of algorithms dealing with visual input data (Lange & Riedmiller, 2010; Yarats et al., 2021). Autoenconders consist of three main components, namely:- an encoder $f_\phi : \mathcal{X} \to \mathcal{Z}$, a latent space representation $z \in \mathcal{Z}$, and a decoder $f_\theta : \mathcal{Z} \to \mathcal{X}$. The encoder compresses the input vector $\mathbf{x}$ into a lower-dimensional representation $\mathbf{z}$, known as the latent variable, while the decoder decodes $\mathbf{z}$ to reconstruct $\tilde{\mathbf{x}}$ (Michelucci, 2022).

Variational Inference. Variational inference is a framework for approximating complex probability densities (Ganguly & Earp, 2021). It entails using a metric (such

as Kullback–Leibler (KL) divergence (Cover & Thomas, 2006)) to choose a tractable estimate of the *posterior* probability density $p(\mathbf{z} \mid \mathbf{x})$, defined using Bayes' Theorem as

$$p(\mathbf{z} \mid \mathbf{x}) \triangleq \frac{p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})}{p(\mathbf{x})} \tag{2.5}$$

where $\mathbf{x}$ and $\mathbf{z}$ are respectively observed and latent variables. $p(\mathbf{x} \mid \mathbf{z})$ is the *likelihood*, $p(\mathbf{z})$ is the *prior*, and $p(\mathbf{x})$ is the *evidence*. To approximate $p(\mathbf{z} \mid \mathbf{x})$, a probability density is selected from a family of tractable probability densities $q(\mathbf{z} \mid \mathbf{x}) \in \mathcal{Q}$. The best candidate q is the one with minimum KL-divergence (Blei et al., 2017), i.e.

$$q^* = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}\left(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z} \mid \mathbf{x})\right) \tag{2.6}$$

where $\mathrm{KL}\left(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z} \mid \mathbf{x})\right)$ is the KL-divergence representing the discrepancy between q and the posterior. Equation 2.6 simplifies to (Ganguly & Earp, 2021):

$$q^* = \arg\min_{q \in \mathcal{Q}} \left\{\mathrm{KL}\left(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})\right) - \mathbb{E}_q[\log p(\mathbf{x} \mid \mathbf{z})]\right\} \tag{2.7}$$

and serves as the basis for *variational autoencoders* (VAEs).

Variational autoencoder (VAE). VAE is a type of an autoencoder that employs variational inference to estimate probability densities and generate data from them (Doersch, 2016). Referring to Figure 2.3 and Equation 2.7, a VAE consists of an encoder that learns parameters μ_ϕ and σ_ϕ for $q(\mathbf{z} \mid \mathbf{x})$ — which is selected to be a Gaussian distribution, i.e. $q = \mathcal{N}(\mu_\phi, \sigma_\phi)$. The prior is defined as a normal distribution $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, and the decoder estimates the likelihood, i.e. $f_\theta(\mathbf{z}) \approx p(\mathbf{x} \mid \mathbf{z})$. By defining the empirical error (loss function) as

$$\mathcal{L} = \mathrm{KL}\left(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})\right) - \mathbb{E}_q[\log p(\mathbf{x} \mid \mathbf{z})] \tag{2.8}$$

and minimising it using *backpropagation*, determines the encoder and decoder parameters ϕ and θ, respectively, that output q which satisfies Equation 2.6.

During inference, latent samples can be easily generated from a normal distribution, i.e. $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and converted (using the decoder $f_\theta(\mathbf{z})$) into data samples $\tilde{\mathbf{x}}$ which are similar to the original data $\mathbf{x} \sim p(\mathbf{x})$. For this reason, VAEs are classified as *generative models* (Ren, 2022), moreover, they are capable of approximating complex probability densities including densities for high-dimensional continuous data (Doersch, 2016). The main drawback of VAEs is that they cannot estimate conditional probabilities densities. This inspired the creation of *conditional variational autoencoders* (cVAEs).

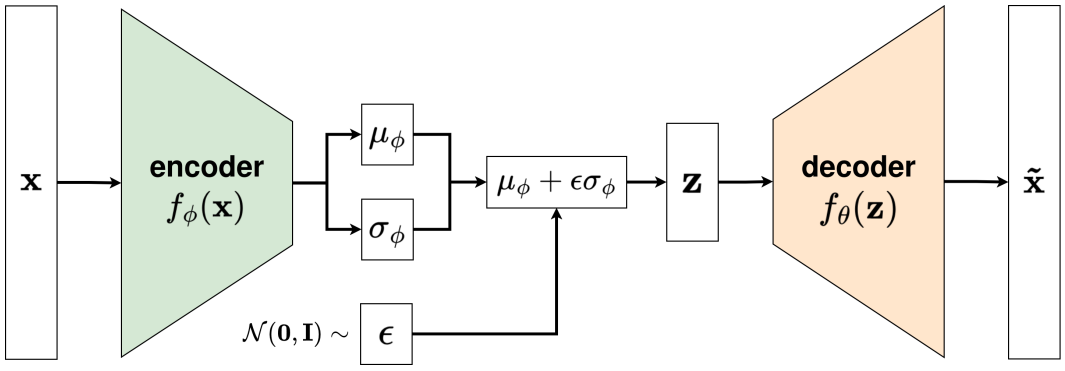


Figure 2.3: *Schematic of the structure of a variational autoencoder. The encoder that takes* $\mathbf{x}$ *and produces parameters for a distribution* $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu_\phi, \sigma_\phi)$ *over the latent space.* $\mathbf{z}$ *samples are decoded such that* $\tilde{\mathbf{x}} \sim p_\theta(\mathbf{x} \mid \mathbf{z}) = f_\theta(\mathbf{z})$.

Conditional variational autoencoder (cVAE). cVAEs are an extension of VAEs that can approximate complex conditional probability densities (Pagnoni et al., 2018). Their architecture is illustrated in Figure 2.4. In cVAEs, the posterior is

$$p(\mathbf{z} \mid \mathbf{x}, \mathbf{y}) = \frac{p(\mathbf{x} \mid \mathbf{y}, \mathbf{z})p(\mathbf{z} \mid \mathbf{y})}{p(\mathbf{x} \mid \mathbf{y})} \tag{2.9}$$

where $\mathbf{y}$ is the conditional variable, $\mathbf{x}$ is the observed variable and $\mathbf{z}$ is the latent variable. The encoder learns parameters for its estimate, i.e. $q(\mathbf{z} \mid \mathbf{x}, \mathbf{y}) = \mathcal{N}(\mu_\phi, \sigma_\phi)$. The prior is defined as a normal distribution $p(\mathbf{z} \mid \mathbf{y}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, while the decoder approximates the likelihood $f_\theta(\mathbf{y}, \mathbf{z}) \approx p(\mathbf{x} \mid \mathbf{y}, \mathbf{z})$. cVAEs are trained by minimising the loss function:

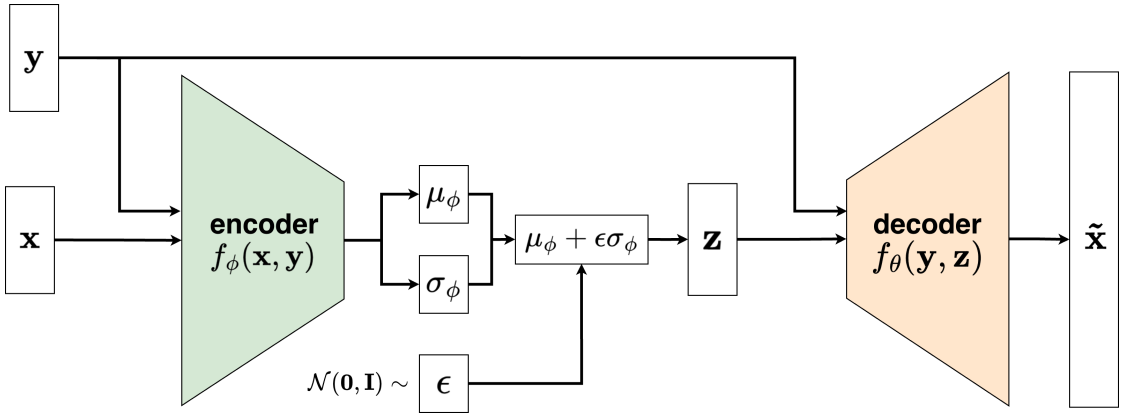


Figure 2.4: *Schematic of the structure of a conditional variational autoencoder. The encoder that takes* $\mathbf{x}, \mathbf{y}$ *and produces parameters for a distribution* $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y}) = \mathcal{N}(\mu_\phi, \sigma_\phi)$ *over the latent space.* $\mathbf{z}$ *samples are decoded such that* $\tilde{\mathbf{x}} \sim p_\theta(\mathbf{x} \mid \mathbf{y}, \mathbf{z}) = f_\theta(\mathbf{y}, \mathbf{z})$

.

$$\mathcal{L} = \mathrm{KL}\left(q(\mathbf{z} \mid \mathbf{x}, \mathbf{y}) \parallel p(\mathbf{z} \mid \mathbf{y})\right) - \mathbb{E}_q[\log p(\mathbf{x} \mid \mathbf{y}, \mathbf{z})] \tag{2.10}$$

Once the optimal encoder parameters ϕ and decoder parameters θ are determined, latent samples can be generated from a normal distribution, i.e. $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and be combined with the condition $\mathbf{y}$ to produce data samples $\tilde{\mathbf{x}}$ via the decoder $f_\theta(\mathbf{y}, \mathbf{z})$. These samples $\tilde{\mathbf{x}}$ are similar to those originating from $p(\mathbf{x} \mid \mathbf{y})$. This makes cVAEs powerful probabilistic models that can estimate complex conditional probability densities, and are suitable for high-dimensional continuous data (Sohn et al., 2015).

In this section, we have provided a description of NN and their applications. These make an integral part of modern RL methods. In the subsequent section, we will discuss the taxonomy of RL algorithms.

2.4 Taxonomy of RL Algorithms

In RL, algorithms solve the MDP by learning the optimal policy and/or its corresponding value functions (Morales et al., 2021). The algorithms pursue this objective via the use of various methods. A taxonomy of RL algorithms, based on works by Zhang & Yu (2020) and Tan et al. (2025), is discussed in this section to provide clarity. An overview of the taxonomy, which is limited to some of the algorithms employed in this dissertation, is presented in Figure 2.5.

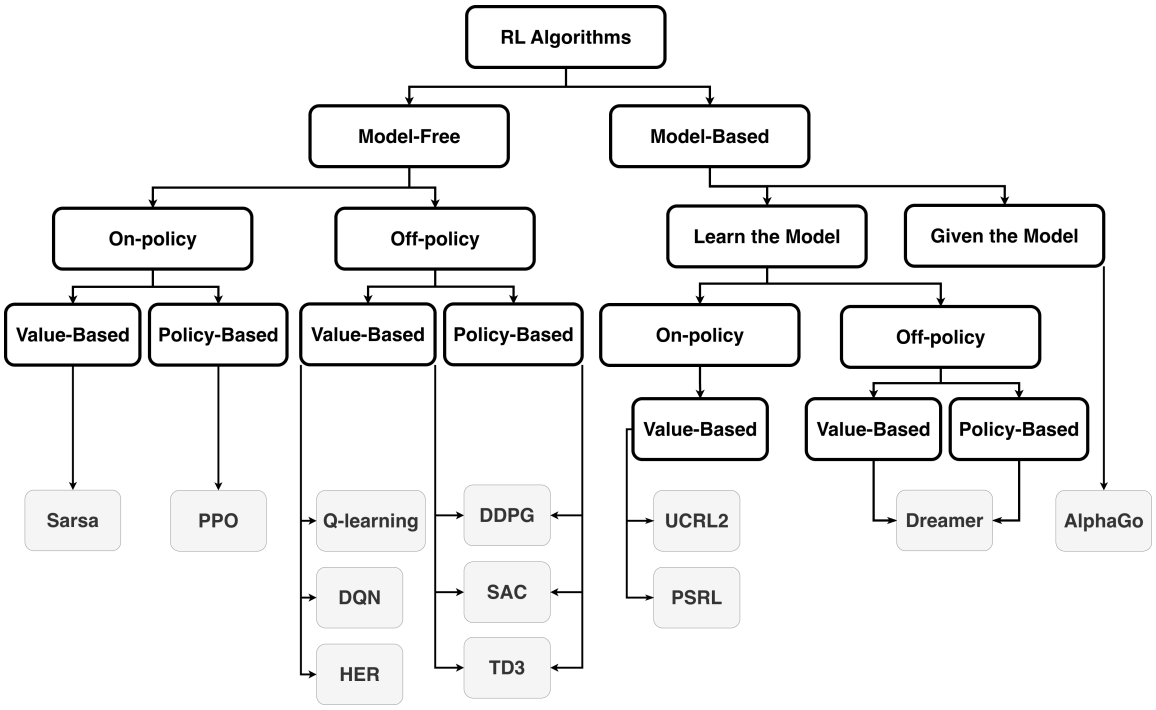


Figure 2.5: *An non-exhaustive overview of RL algorithms. Boxes with thick boundaries denote categories, while the rest present specific algorithms.*

Model-based RL (MBRL) and Model-free RL (MFRL). As previously discussed in Section 1.2, MBRL methods require an explicit dynamic model of the environment (i.e. predictive model) to learn a policy and/or value function, while MFRL approaches do not. In MBRL, algorithms learn through using the predictive model to simulate trajectories. MFRL algorithms learn through direct exploration of the environment (Huang, 2020; Tan et al., 2025). In non-tabular settings, both methods employ neural networks, to approximate either the dynamic model, policy or value function (Bertsekas & Tsitsiklis, 1995; Spielberg et al., 2017; Sutton & Barto, 2018). Note that MBRL algorithms are classified as *given the model* (e.g. AlphaGo (Silver et al., 2017)) and *learn the model* (e.g. Dreamer (Hafner et al., 2020, 2025)), prior using similar classifications for MFRL algorithms which include *on-policy* and *off-policy* (Tan et al., 2025).

On-policy and Off-policy. On-policy and off-policy learning involves the way algorithms train to improve the policy or value function (Wang et al., 2024). In on-policy learning, algorithms use the policy to generate the training data while simultaneously improving it (Sutton & Barto, 2018). The training data is newly generated and discarded after each round of optimisation (Wang et al., 2024). In off-policy learning, algorithms utilise a *behaviour policy* to generate training samples which are stored in a buffer, and later used to improve a *learning policy* (Sutton et al., 2011). The off-policy approach is more data efficient than the on-policy approach; however unlike its counterpart, it uses out-of-date experiences (stored) which may lead to learning instabilities (van Hasselt et al., 2018; Fakoor et al., 2020). Within on-policy and off-policy groups, there are *value-based* and *policy-based* classes.

Value-Based and Policy-Based. The value-based approach consists of approximating a value function and using it to compute a policy (Sutton et al., 1999a). The policy is implicitly denoted as $\pi = \arg\max_{a \in \mathcal{A}} Q^*(s, a)$, and it chooses the action in each state with the highest estimated value (Sutton & Barto, 2018). This approach is simpler and more sample efficient than its counterpart (Arulkumaran et al., 2017; Sutton & Barto, 2018). Nevertheless, it requires discretisation of continuous action spaces (Wang et al., 2024) and is limited to discovering deterministic policies (Sutton et al., 1999a).

The policy-based method entails approximating a policy directly, where the policy is represented using a function approximator (Sutton & Barto, 2018). The method outputs stochastic policies, which do not require discretisation and hence suitable for both discrete and continuous action spaces (Wang et al., 2024). The disadvantage is that the approach has high-variance gradient estimates (V. Konda & Tsitsiklis, 1999) which lead to slow and sample inefficiency learning (Arulkumaran et al., 2017; Sutton & Barto, 2018)

Actor-critic method is a combination of value-based and policy-based approaches (Sutton & Barto, 2018; Morales et al., 2021). The method approximates both the policy and value function. The actor is the policy model and it selects actions. The critic is the value function and it evaluates the actions of the actor. Both actor and critic

have independent parameters. The actor-critic method has the advantages of both the policy-based and value-based approaches and is widely employed by many SoTA RL algorithms (Arulkumaran et al., 2017).

In this section, we briefly explored the taxonomy of RL algorithms, which is helpful in understanding the core approaches of most algorithms. In the subsequent section, we will look further into how some RL algorithms function.

2.5 RL Algorithms

This section discusses some of the algorithms employed in the thesis work (shown in Figure 2.5). These include Q-learning (Watkins & Dayan, 1992), UCRL2 (Jaksch et al., 2010), PSRL (Osband et al., 2013) and DQN (Mnih et al., 2013), which are limited to discrete action spaces. Additionally, DDPG (Lillicrap et al., 2016), SAC (Haarnoja et al., 2018) and TD3 (Fujimoto et al., 2018) are included — which are used in continuous action spaces. These algorithms are selected because they *are state-of-the-art* (DDPG, SAC and TD3), *have theoretical bounds* (UCRL2 and PSRL), and *are popular* (Q-learning and DQN).

In general, most RL algorithms aim to learn a function f in an incremental way, often using the difference between the target value and the estimated value of the function as $f_{new} \leftarrow f_{old} + \eta(f_{target} - f_{old})$ (Morales et al., 2021). Note that $\eta \in (0, 1]$ is the learning rate parameter that manages the rate at which f_{new} is updated (Sutton & Barto, 2018). It should be intricately adjusted to allow sufficient updating without overshooting the estimate. The function f can refer to a policy and/or value function depending on the algorithm. Often f_{target} is unknown and has to be estimated. Algorithms can be distinguished based on *what they denote* f_{target} *to be* and *how they estimate it*, as shown in this section.

2.5.1 Q-learning

Q-learning is a classic value-based algorithm (Wang et al., 2024) that was introduced by Watkins & Dayan (1992). It is a simple-to-implement algorithm that functions well in tabular settings. It uses a lookup table to store state-action values, and relies on learning the optimal Q-value function $Q^*(s, a)$ to find an optimal policy (François-Lavet et al., 2018). The algorithm learns the Q-value function in an incremental way using (Sutton & Barto, 2018),

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \eta \left(\underbrace{\mathcal{R}(s_t, a_t) + \gamma \max_{a \in \mathcal{A}} Q_t(s_{t+1}, a)}_{Q(s_t, a_t)_{target}} - Q_t(s_t, a_t) \right) \tag{2.11}$$

where η is the learning rate, $\mathcal{R}(s_t, a_t)$ is the reward issued at state-action pair (s_t, a_t), $Q_t(s_t, a_t)$ is the value of (s_t, a_t) at timestep t, and $Q(s_t, a_t)_{target}$ is estimated. The

algorithm is model-free and off-policy. It uses a behavioural policy called $\epsilon-$*greedy strategy* that normally selects actions with the highest Q-values, but with a small ϵ probability it sometimes randomly chooses actions, i.e.

$$\begin{aligned} a &= \arg\max_{a \in \mathcal{A}} Q_t(s,a), \quad \text{with probability } 1-\epsilon \\ a &\sim \mathrm{U}(\min \mathcal{A}, \max \mathcal{A}), \quad \text{with probability } \epsilon \end{aligned} \tag{2.12}$$

where $\epsilon \in [0,1]$ is a hyperparameter that manages the state-action space exploration during the learning process via selection of random actions.

For the Q-value to converge to $Q^*(s,a)$, all state-action pairs in the state-action space must be continually visited (Sutton & Barto, 2018). This limits the application of the algorithm to spaces with low dimensionality. To circumvent this limitation, neural networks were introduced to substitute the lookup table (Sutton & Barto, 2018). This enables the Q-value function to generalise over the state-action space without necessary visiting all the state-action pairs (François-Lavet et al., 2018; Moerland et al., 2023). By combining Q-learning with neural networks, this approach gave rise to an algorithm called DQN.

2.5.2 Deep Q-Network (DQN)

DQN uses neural networks to represent the value function $Q_\theta(s,a)$, where θ are parameters that define Q-values. It is a pioneer algorithm that set the precedent for attaining above human-level performance in non-tabular settings. The algorithm was introduced by Mnih et al. (2013) to extract low-level features from raw images of video games and approximate the Q-value function without domain knowledge (Wang et al., 2024). Similar to Q-learning, it utilises $\epsilon-$greedy strategy for exploration (François-Lavet et al., 2018), and has a target value at timestep t defined as:

$$Y_\theta = \mathcal{R}(s_t, a_t) + \gamma \max_{a \in \mathcal{A}} Q_\theta(s_{t+1}, a) \tag{2.13}$$

DQN uses a buffer (or *experience replay*) to store trajectories obtained by the behavioural policy. During training, samples from the buffer are randomly and uniformly selected to update parameters θ through minimising the loss function:

$$L(\theta) = \mathbb{E}_{(s_i, a_i, \mathcal{R}(s_i,a_i), s_{i+1}) \sim B} \left[\left(Y_{\theta^-} - Q_\theta(s,a) \right)^2 \right] \tag{2.14}$$

via *backpropagation*. Note that $(s_i, a_i, \mathcal{R}(s_i,a_i), s_{i+1}) \sim B$ denotes data samples from the buffer, and Y_{θ^-} is the target value from a replica of Q_θ (i.e. Q_{θ^-}), whose parameters θ^- have lagging updates. The reason the parameters θ^- update slower than θ is to improve the stability of the algorithm (Mnih et al., 2015). Q_{θ^-} is referred to as the *target network*.

Similar to Q-learning, DQN only addresses problems with discrete actions (Lillicrap et al., 2016). A common approach when deploying DQN in environments with continuous actions is to discretize them. However, discretization can result in exploding number of discrete actions or insufficient exploration of the action space, and non-smooth control outcomes (Lillicrap et al., 2016; Wang et al., 2024). To circumvent this, an algorithm inspired by DQN with an *actor-critic* architecture called DDPG was introduced.

2.5.3 Deep deterministic policy gradient (DDPG)

DDPG learns both the policy and Q-value function simultaneously. Both are represented by neural network architectures as π_ϕ and Q_θ, respectively. It is a state-of-the-art (SoTA) algorithm in non-tabular settings (Lazaridis et al., 2020). The algorithm was established by Lillicrap et al. (2016) with the aim of deploying it in robotic tasks. It is model-free, off-policy, and employs the experience replay buffer and target networks (to stabilise learning). The algorithm adds noise to its policy during exploration as follows,

$$a_t = \pi_\phi(s_t) + \beta, \quad \text{where } \beta \sim \mathcal{N} \tag{2.15}$$

where $\mathcal{N}$ is Gaussian noise, however can be replaced by other random noise generators. The target value at each timestep is:

$$Y_\theta = \mathcal{R}(s_t, a_t) + \gamma Q_{\theta^-}(s_{t+1}, \pi_{\phi^-}(s_{t+1})) \tag{2.16}$$

while parameters θ and ϕ are updated through minimising the loss functions

$$L_1(\theta) = \mathbb{E}_{(s_i, a_i, \mathcal{R}(s_i, a_i), s_{i+1}) \sim B} \left[\left(Y_{\theta^-} - Q_\theta(s, a) \right)^2 \right] \tag{2.17}$$

$$L_2(\phi) = -\mathbb{E}_{s_i \sim B} \left[Q_\theta(s, \pi_\phi(s)) \right] \tag{2.18}$$

Note that both the policy and Q-value function have their respective target networks π_{ϕ^-} and Q_{θ^-} with lagging updates. While DDPG was an improvement of DQN, its performance is however highly sensitive to hyperparameter settings (Haarnoja et al., 2018; Wang et al., 2022; Adkins et al., 2024). This makes the algorithm less robust. To address this shortcoming, the SAC and TD3 algorithms (which are based on DDPG) were developed.

2.5.4 Soft actor critic (SAC)

SAC (Haarnoja et al., 2018) is a model-free, off-policy, actor-critic deep RL algorithm based on maximising the entropy of the policy's action distribution. SAC is a SoTA

algorithm in non-tabular settings as well (Lazaridis et al., 2020). It has been deployed extensively in robotic tasks (Beltran-Hernandez et al., 2020b,a; Ibarz et al., 2021; Zhang et al., 2023; Beltran-Hernandez et al., 2020a). While DDPG uses a deterministic policy, which requires additional noise during exploration, SAC employs a stochastic policy. This makes SAC have superior exploration capabilities and often better performance.

Suppose one is given a stochastic policy $\pi_\phi(a \mid s)$, then the entropy over the policy distribution for each state in SAC is

$$\mathbb{H}(\pi_\phi(\cdot \mid s_t)) = -\mathbb{E}_{a_t \sim \pi_\phi}\left[\log \pi_\phi(a_t \mid s_t)\right] \tag{2.19}$$

$\mathbb{H}(\pi_\phi(\cdot \mid s_t))$ is added to the Q-value function as an entropy bonus as follows:

$$Q_\theta(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \left(\mathcal{R}(s_t,a_t) + \alpha \mathbb{H}(\pi_\phi(\cdot \mid s_t))\right) \mid s_0 = s, a_0 = a\right] \tag{2.20}$$

where α coefficient is a parameter that manages the stochasticity of the policy (Haarnoja et al., 2019). This approach incentivises the policy to explore the action space more broadly as witnessed in (Haarnoja et al., 2017; Schulman et al., 2017a). Unlike DDPG, SAC employs two Q-value functions along with corresponding target networks to improve the learning stability. Note that the target networks are only for the Q-value functions. SAC's target value is thus (Haarnoja et al., 2018, 2019),

$$Y_\theta = \mathcal{R}(s_t,a_t) + \gamma\left(\min_{n=1,2} Q_{\theta_n^-}(s_{t+1},a_{t+1}) - \alpha \log \pi_\phi(a_{t+1} \mid s_{t+1})\right) \tag{2.21}$$

where $a_{t+1} \sim \pi_\phi(\cdot \mid s_{t+1})$. It should be noted that the policy parameters ϕ and Q-value functions parameters θ_1, θ_2 are updated using the loss functions:

$$L_1(\theta_n) = \mathbb{E}_{(s_i,a_i,\mathcal{R}(s_i,a_i),s_{i+1})\sim B}\left[\left(Y_{\theta^-} - Q_{\theta_n}(s,a)\right)^2\right], \quad \text{for } n = 1,2 \tag{2.22}$$

$$L_2(\phi) = -\mathbb{E}_{s_i \sim B}\left[\min_{n=1,2} Q_{\theta_n}(s, \tilde{a}_\phi(s)) - \alpha \log \pi_\phi(\tilde{a}_\phi(s) \mid s)\right], \text{ where } \tilde{a}_\phi(s) \sim \pi_\phi(\cdot \mid s) \tag{2.23}$$

2.5.5 Twin Delayed DDPG (TD3)

TD3 (Fujimoto et al., 2018) is a model-free, off-policy, actor-critic deep RL algorithm that is closely similar to DDPG. A common cause of failure in the DDPG algorithm is that the learned Q-value function dramatically overestimates Q-values (Thrun & Schwartz, 2014; Fujimoto et al., 2018). This leads to poor performance since the policy

π_ϕ learns to exploit the errors in the Q-value function. TD3 overcomes this issue by (1) utilising two Q-value functions (similar to SAC), (2) updating its policy π_ϕ model less frequently than the Q-value functions, and (3) smoothening its Q-value function landscape to prevent its policy from exploiting incorrect peaks and discontinuities for some actions (Fujimoto et al., 2018; Lazaridis et al., 2020) as follows:

Suppose the target value for TD3 at each timestep is:

$$Y_\theta = \mathcal{R}(s_t, a_t) + \gamma \mathbb{E}_\epsilon \left[\tilde{Q}_{\theta^-}(s_{t+1}, \pi_{\phi^-}(s_{t+1}) + \epsilon) \right] \tag{2.24}$$

where $\tilde{Q}_{\theta^-}(\cdot, \cdot) = \min_{n=1,2} Q_{\theta_n^-}(\cdot, \cdot)$ is the Q-value function with the smaller state-action value and ϵ is noise added to the target action $a_{t+1} = \pi_{\phi^-}(s_{t+1})$. Adding noise to the target action makes the Q-value depend not on a single action but on a local neighbourhood of actions around a_{t+1}. This process improves the learning stability and is often referred to as *target policy smoothing regularisation* (Fujimoto et al., 2018).

TD3 is also a SoTA algorithm in non-tabular settings (Lazaridis et al., 2020) and has been used in several robotic tasks (Hou et al., 2021; Fernandez-Fernandez et al., 2022). It updates its policy parameters ϕ similarly to DDPG and its Q-value functions parameters θ_1, θ_2 similarly to SAC via the minimisation of:

$$L_1(\theta_n) = \mathbb{E}_{(s_i, a_i, \mathcal{R}(s_i, a_i), s_{i+1}) \sim B} \left[\left(Y_{\theta^-} - \tilde{Q}_\theta(s, a) \right)^2 \right], \quad \text{for } n = 1, 2 \tag{2.25}$$

$$L_2(\phi) = -\mathbb{E}_{s_i \sim B} \left[\tilde{Q}_\theta(s, \pi_\phi(s)) \right] \tag{2.26}$$

2.5.6 Hindsight Experience Replay (HER)

Although the development of DDPG, SAC and TD3 enjoys success in many problems with continuous action spaces (Lazaridis et al., 2020), it often struggles to perform well in robotic tasks with sparse reward signals (Andrychowicz et al., 2017). These tasks make it challenging for the algorithms to find (near-)optimal policies due to insufficient feedback reward signals during learning. However, it is often desirable to maintain sparse rewards in these tasks to avoid the shortcomings of reward shaping (as discussed in Sections 1.2 and 2.2.3).

HER is a technique established by Andrychowicz et al. (2017) that can be combined with off-policy algorithms to enhance their learning capabilities in sparse-rewards settings. It achieves this by pairing trajectories $h_{t:T} = (s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}, s_T)$ with goal states g, and reconceptualising failed trajectories as successful ones for alternative goals g'. For instance, if an agent experienced $h_{t:T}$ while attempting to reach the intended goal state g but reached some state $s_T = s' \neq g$; then $h_{t:T}$ is reimagined as aiming to achieve the alternative goal $g' = s'$. With this, the agent learned the sequence of actions that led to attaining the goal state g'. Consequently, the agent re-

ceives informative feedback over the set of goals $\{g_i'\}_{i\geq 1}$ using unsuccessful trajectories. After training, if $g \in \{g_i'\}_{i\geq 1}$ then the agent performs the task successfully.

2.5.7 Upper confidence bounds for reinforcement learning (UCRL2)

UCRL2 is a model-based, on-policy and value-based algorithm that was introduced by Jaksch et al. (2010). UCRL2 has strong theoretical basis and is provably regret-optimal. Studying it can help connect theory with approaches for assessing algorithms. It is assumes the MDP $\mathbb{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T})$ is unknown and has to be learned through empirically estimating the rewards $\hat{\mathcal{R}}_k(s,a)$ and transition probabilities $\hat{T}_k(s' \mid s,a)$ from past experience using,

$$\hat{\mathcal{R}}_k(s,a) = \frac{1}{n(s,a)} \sum_{i=1}^{n(s,a)} \mathcal{R}_i(s,a)\,, \tag{2.27}$$

$$\hat{T}_k(s' \mid s,a) = \frac{1}{n(s,a)} \sum_{i=1}^{n(s,a)} \mathbb{I}\{s' \text{ observed after } (s,a)\}\,, \tag{2.28}$$

where $n(s,a)$ is the number of visits to the state-action pair (s,a), $k \in \mathbb{N}$ is the number of episodes, and $\mathbb{I}$ is an indicator function. This means an MDP $\tilde{\mathbb{M}}_k = (\mathcal{S}, \mathcal{A}, \hat{\mathcal{R}}_k, \hat{\mathcal{T}}_k)$ approximates $\mathbb{M}$ with some uncertainty, where $\hat{\mathcal{R}}_k$ and $\hat{\mathcal{T}}_k$ are MDP parameters. However, the optimal policy of $\tilde{\mathbb{M}}_k$, i.e. $\pi^*_{\tilde{\mathbb{M}}_k}$, may be misleading to deploy due to estimation errors (Filippi et al., 2010). To tackle this issue, the algorithm maintains a set of statistically plausible MDPs $\mathcal{M}_{k,\delta} = \{\tilde{\mathbb{M}}^{(1)}, \tilde{\mathbb{M}}^{(2)}, \ldots, \tilde{\mathbb{M}}^{(n)}\}$ including $\tilde{\mathbb{M}}_k$, whose rewards and transition probabilities have confidence intervals,

$$|\mathcal{R}(s,a) - \hat{\mathcal{R}}_k(s,a)| \leq \sqrt{\frac{3.5 \log(\frac{2|\mathcal{S}||\mathcal{A}|k}{\delta})}{n(s,a)}} = \epsilon_{\mathcal{R}} \tag{2.29}$$

$$\|T(\cdot \mid s,a) - \hat{T}_k(\cdot \mid s,a)\|_1 \leq \sqrt{\frac{14|\mathcal{S}| \log(\frac{2|\mathcal{A}|k}{\delta})}{n(s,a)}} = \epsilon_{\mathcal{T}} \tag{2.30}$$

To balance the exploration-exploitation trade-off, UCRL2 uses the principle of *optimism in the face of uncertainty* (OFU). That is, the agent always try out a policy that stands a chance of being highly profitable (Osband & Roy, 2017; Pacchiano et al., 2021). By executing this policy (called *optimistic policy*), the agent either gains high rewards or information about the true MDP (Wu et al., 2022). From $\mathcal{M}_{k,\delta}$, UCRL2 selects an MDP $\tilde{\mathbb{M}}$ that leads to the highest cumulative rewards, called the *optimistic MDP*. The optimistic policy is thus derived from the optimistic MDP as $\pi_k = \arg\max_\pi \left[\max\left(J^\pi(\tilde{\mathbb{M}}) \mid \tilde{\mathbb{M}} \in \mathcal{M}_{k,\delta}\right)\right]$ (Bourel et al., 2020).

To determine the optimal policy π^* for the true MDP, the state value function $V(s)$ is updated using (Jaksch et al., 2010):

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \left\{ \left(\hat{\mathcal{R}}_k(s,a) + \epsilon_{\mathcal{R}} \right) + \max_{T \in \mathcal{P}(s,a)} \sum_{s'} T(s' \mid s,a) \cdot V(s') \right\} \tag{2.31}$$

where $\mathcal{P}(s,a)$ is a set of transition distributions with bounds defined in Equation 2.30. Once $V(s)$ has converged to $V^*(s)$, the optimal policy is given by:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left\{ \left(\hat{\mathcal{R}}_k(s,a) + \epsilon_{\mathcal{R}} \right) + \max_{T \in \mathcal{P}(s,a)} \sum_{s'} T(s' \mid s,a) \cdot V^*(s') \right\} \tag{2.32}$$

Note that UCRL2 updates its policy model episodically rather than every step, and the computation of its confidence intervals limits the algorithm to tabular settings. Nevertheless, the algorithm is theoretically grounded because it has regret bounds. The advantage of OFU in UCRL2 is that it encourages exploration of the state-action space while occasionally being highly rewarding. This significantly improves the agent's certainty about the true MDP.

2.5.8 Posterior sampling for reinforcement learning (PSRL)

PSRL is a model-based, on-policy and value-based algorithm that was introduced by Osband et al. (2013). Unlike UCRL2 that applies OFU to reduce the agent's uncertainty about the true MDP, PSRL employs *posterior sampling*. PSRL is a SoTA RL algorithm in tabular settings with concrete theoretical analysis (Osband et al., 2013). The algorithm starts with a prior distribution over MDPs (specifically MDP parameters $\hat{\mathcal{R}}_k$ and $\hat{\mathcal{T}}_k$), and samples a single statistically plausible MDP $\tilde{\mathbb{M}}_k$ from the posterior distribution conditioned on the trajectory on each k^{th} episode (Osband & Roy, 2017). Afterwards, the agent acts per-episode according to the optimal policy of the sampled MDP $\pi^*_{\tilde{\mathbb{M}}_k}$.

For example, the transition probabilities can be sampled from a prior Dirichlet distribution, while rewards can be sampled from a Gaussian distribution (Strens, 2000) as follows:

$$\hat{\mathcal{R}}_0(s,a) \sim \mathcal{N}(\mu_{s,a}, \sigma^2) \tag{2.33}$$

$$\hat{T}_0(\cdot \mid s,a) \sim \text{Dirichlet}(\alpha_{s,a}) \tag{2.34}$$

where $\alpha_{s,a}$, $\mu_{s,a}$ and σ^2 are parameters. Bayes' Theorem is then applied to update the prior distributions. At the start of each k^{th} episode, $\hat{\mathcal{R}}_k$ and $\hat{\mathcal{T}}_k$ are sampled (which is equivalent to sampling an MDP $\tilde{\mathbb{M}}_k = (\mathcal{S}, \mathcal{A}, \hat{\mathcal{R}}_k, \hat{\mathcal{T}}_k)$) and the policy selected for the episode is $\pi_k = \arg\max_\pi \left[J^\pi(\tilde{\mathbb{M}}_k) \right]$.

Unlike OFU, exploration in PSRL is driven by the variability in the sampled MDPs and thus policies (Osband et al., 2013; Osband & Roy, 2017). That is, if the MDPs are substantially different, the resulting (i.e. corresponding) optimal policies would significantly differ and thus the agent would take diverse actions at every k^{th} episode. This would over time improve the agent's certainty about the state-action space. PSRL is considered more computationally efficient than UCRL2; however, the algorithm is also limited to tabular settings.

2.6 Concluding Remarks

In this chapter, we discussed the basic elements of MDPs and a few RL algorithms. Although the algorithms can be distinct, they have the same approach of solving MDPs through maximising the expected return. The success of algorithms is depended on their *exploration* methods. This is because finding the optimal policies depends on the *exploration-exploitation interplay strategies* of the algorithms (Kaelbling et al., 1996; Sutton & Barto, 2018). Although exploration is important, there remains a lack of consensus over approaches that can quantitatively compare exploration methods across RL algorithms (Seijen et al., 2020; Amin et al., 2021; Ladosz et al., 2022). In the next chapter, we delve into *exploration in RL* and introduce a framework for capturing *exploration efficiency*.

Chapter 3

Studying Exploration in RL: An Optimal Transport Analysis of Occupancy Measure Trajectories

This chapter is taken from the following article (also referred to as (Nkhumise et al., 2025)):

Reabetswe M. Nkhumise, Debabrota Basu, Tony J. Prescott, Aditya Gilra. "Studying Exploration in RL: An Optimal Transport Analysis of Occupancy Measure Trajectories." In Transactions on Machine Learning Research, May 2025. `https://openreview.net/pdf?id=pdC092Nn8N`

3.1 Abstract

The rising successes of RL are propelled by combining smart algorithmic strategies and deep architectures to optimize the distribution of returns and visitations over the state-action space. A quantitative framework to compare the learning processes of these RL algorithms is currently absent but desired in practice. We address this gap by representing the learning process of an RL algorithm as a sequence of policies generated during training, and then studying the policy trajectory induced in the manifold of state-action occupancy measures. Using an optimal transport-based metric, we measure the length of the paths induced by the policy sequence yielded by an RL algorithm between an initial policy and a final optimal policy. Hence, we first define the *Effort of Sequential Learning (ESL)*. ESL quantifies the relative distance that an RL algorithm travels compared to the shortest path from the initial to the optimal policy. Furthermore, we connect the dynamics of policies in the occupancy measure space and regret (another metric to understand the sub-optimality of an RL algorithm), by defining the *Optimal Movement Ratio* (OMR). OMR assesses the fraction of movements in the occupancy measure space that effectively reduce an analogue of regret. Finally, we derive approximation guarantees to estimate ESL and OMR with a finite number

of samples and without access to an optimal policy. Through empirical analyses across various environments and algorithms, we demonstrate that ESL and OMR provide insights into the exploration processes of RL algorithms and the hardness of different tasks in discrete and continuous MDPs.

3.2 Introduction

In recent years, significant advancements in RL have been achieved in developing exploration techniques that improve learning (Bellemare et al., 2016; Burda et al., 2019; Eysenbach et al., 2019) along with new learning methods (Lazaridis et al., 2020; Müller et al., 2021; Li, 2023). However, there remains a lack of consensus over approaches that can quantitatively compare these exploration techniques across RL algorithms and tasks (Seijen et al., 2020; Amin et al., 2021; Ladosz et al., 2022). This is attributed to some methods being algorithm-specific (Tang et al., 2017), while others provide theoretical guarantees for very specific settings (Lattimore & Szepesvári, 2020; Agarwal et al., 2022). Thus, comparing exploration techniques of various algorithms across the multi-directional space of RL algorithm design emerges as a natural question. Nevertheless, the present literature lacks a metric to compare them except regret, which is often hard to estimate (Ramos et al., 2017, 2018).

This chapter aims to address this gap based on two key observations. *First*, we observe from the linear programming formulation of RL that solving the value maximisation problem is equivalent to finding an optimal occupancy measure (Syed et al., 2008; Neu & Pike-Burke, 2020; Kalagarla et al., 2021). Occupancy measure is the distribution of state-action pair visits induced by a policy (Altman, 1999; Laroche & des Combes, 2023). Under mild assumptions, a policy maps uniquely to an occupancy measure. *Second*, we observe that any RL algorithm learns by sequentially updating policies starting from an initial policy to reach an optimal policy. The search for an optimal policy is influenced by the exploration-exploitation strategy and function approximators, both of which impact the overall performance of the agent by determining the quality of experiences from which it learns (Zhang et al., 2019; Ladosz et al., 2022). Hereby, we term collectively the learning strategy and the exploration-exploitation interplay as the *exploratory process*.

Contributions. *1. A Framework.* Motivated by our observations, we abstract any RL algorithm as a trajectory of occupancy measures induced by a sequence of policies between an initial and a final (optimal) policy. The occupancy measure of a policy given an environment corresponds to the data-generating distribution of state-action pairs. Thus, we can quantify the effort of each policy update, i.e. the effort required to shift the state-action data distributions, as the transportation distance between their occupancy measures. The total effort of learning by the algorithm can be measured as the total distance covered by its occupancy measure trajectory. We provide a mathematical basis for this quantification by proving that the space of occupancy measures is a differentiable manifold for smoothly parameterized policies (Section 3.4). Hence, we can compute the length of the occupancy measure trajectory on this manifold using

Wasserestein distance as the metric (Villani, 2008).

2. Effort of Sequential Learning. In contrast to RL, if we knew the optimal policy we could update our initial policy directly via supervised or imitation learning. Effort of this learning is represented by a direct, shortest (geodesic) path from initial to optimal policy on the occupancy measure manifold. To quantify the cost of the exploratory process to learn the environment, we define the *Effort of Sequential Learning* (ESL) as the ratio of the (indirect) path traversed by an RL algorithm in the occupancy measure space to the direct distance between the initial and optimal policy (Section 3.4.1). Lower ESL implies a more efficient exploratory process.

3. Efforts to learn that lead to Regret-analogue minimisation. Regret is a widely used optimality measure for reward-maximising RL algorithms (Jaksch et al., 2010). It measures the total deviation in the value functions achieved by a sequence of policies learned by an RL algorithm with respect to the optimal algorithm that always uses the optimal policy (Sinclair et al., 2023). We show that regret is related to the sum of distances between the optimal policy and each policy in the sequence learned by the RL algorithm, in the occupancy measure space. We can define an analogue of instantaneous regret (at any one step during learning rather than cumulative), in the occupancy measure space, as the geodesic distance between the occupancy measure of the policy at the current step in the learning sequence, and the optimal one. We find that not all policy updates lead to a reduction in this analogue of immediate regret, and thus define another index *Optimal Movement Ratio* (OMR) that measures the fraction that do (Section 3.4.2).

4. Computational and Numerical Insights. We prove sample complexity guarantees to approximate ESL and OMR in practice as we do not have access to the occupancy measures but collection of roll-outs from the corresponding policies (Section 3.5). We show the relation of empirical OMR and ESL to the true ones if the optimal policy is never reached by an algorithm. We conduct experiments on multiple environments, both discrete and continuous, with sparse and dense rewards, comparing SoTA algorithms. We observe that by visualising aspects of the path traversed (and by comparing ESL and OMR), we are able to compare and provide insights into their exploratory processes and the impact of task hardness on them (Section 3.6).

3.3 Preliminaries

Markov Decision Processes. As previously discussed in Sections 1.2 and 2.2, we consider an agent interacting with an environment in discrete timesteps. At each timestep $t \in \mathbb{N}$, the agent observes a state s_t, executes an action a_t, and receives a scalar reward $\mathcal{R}(s_t, a_t)$. The behaviour of the agent is defined by a stationary Markov policy $\pi(a_t|s_t)$. The environment is modelled as an MDP $\mathbb{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}$ is transition dynamics, $\mathcal{R}$ is the reward function, and γ is the discount factor. $\mathbb{M}$ has an infinite-horizon with the state-value function given by $V_\pi(s) \triangleq \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \mid s_0 = s\right]$ (as formerly discussed in Section 2.2.2).

The goal is to learn a policy that maximises the objective $J_\mu^\pi = \mathbb{E}_{s\sim\mu}[V_\pi(s)]$.

Occupancy Measure. The state-action occupancy measure is a distribution over the $\mathcal{S} \times \mathcal{A}$ space that represents the discounted frequency of visits to each state-action pair when executing a policy π in the environment (Syed et al., 2008; Karwowski et al., 2024). Formally, the occupancy measure of π is

$$v_\pi(s,a) \triangleq \rho \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s, a_t = a \mid \pi, \mu) \tag{3.1}$$

where $\rho = 1 - \gamma$ is the normalizing factor. Stationary Markovian policies allow a bijective correspondence with their state-action occupancy measures (Givchi, 2021). We express the objective J_μ^π in terms of the occupancy measure as

$$J_\mu^\pi = \frac{1}{\rho} \mathbb{E}_{(s,a)\sim v_\pi} \left[\bar{\mathcal{R}}(s,a)\right] \tag{3.2}$$

where $\bar{\mathcal{R}}(s,a) = \mathbb{E}_\pi \left[\mathcal{R}(s_t, a_t) \mid s_t = s, a_t = a\right]$ is the expected immediate reward for the state-action pair (s,a).

Wasserstein Distance. Let $\mu, \nu \in \mathcal{P}(\mathcal{X})$ be probability measures on a complete and separable metric (Polish) space $(\mathcal{X}, d_\mathcal{X})$. The p-Wasserstein distance between μ and ν is (Villani, 2008)

$$\mathcal{W}_p(\mu,\nu) \triangleq \left(\min_{\pi \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{X}} c(x,x')\, d\pi(x,x') \right)^{1/p}, \tag{3.3}$$

where the cost function is given by the metric as $c(x,x') = (d_\mathcal{X}(x,x'))^p$ for some $p \geq 1$. $\Pi(\mu,\nu)$ is a set of all admissible transport plans between μ and ν, i.e. probability measures on $\mathcal{X} \times \mathcal{X}$ space with marginals μ and ν. The Wasserstein distance $\mathcal{W}_p(\mu,\nu)$ is metric that quantifies the distance between probability measures μ, ν. It achieves this by taking a distance on the support space $\mathcal{X}$ and creating out of it a distance on the probability measure space $\mathcal{P}(\mathcal{X})$(Peyré, 2019). With this, we can define a Wasserstein space as follows:

Definition 1 (Wasserstein Space)**.** *(Santambrogio, 2015) Given a Polish space $\mathcal{X}$, for each $p \in [1,\infty)$, the space $\mathcal{P}(\mathcal{X})$ endowed with the distance $\mathcal{W}_p$ is a Wasserstein space $\mathbb{W}_p$ of order p.*

Theorem 5.27 in (Santambrogio, 2015) states that if $\mathcal{X}$ is a convex space, then the space $\mathbb{W}_p$ is a geodesic space (length space). Therefore, the geodesic distance (i.e. shortest path distance on the surface of $\mathcal{P}(\mathcal{X})$) between $\mu, \nu \in \mathcal{P}(\mathcal{X})$ is given by $\mathcal{W}_p(\mu,\nu)$

(Kolouri et al., 2017). Note that the RL problems we consider, consist of the state-action spaces $\mathcal{Z} = \mathcal{S} \times \mathcal{A} \in \mathbb{R}^B : B \geq 2$ that are subsets of the Euclidean space — which is a convex space by definition (Boyd & Vandenberghe, 2004).

Compared to the Kullback-Leibler divergence, or KL-divergence, the Wasserstein distance is a metric that satisfies symmetry and triangle inequality, while KL-divergence is not a metric. $\mathcal{W}_p$ can capture distances between distributions with disjoint support due to leveraging the geometry of the underlying support space $\mathcal{X}$ (Peyré, 2019), while KL-divergence cannot. For this work, we consider 1-Wasserstein distance, i.e. $p = 1$, since $\mathcal{W}_1$ is less sensitive to outliers and sampling discrepancies than $\mathcal{W}_{p>1}$ (Raghvendra et al., 2024). Nevertheless, our results are generalisable to $\mathcal{W}_{p>1}$. Note that $\mathcal{W}_1(\mu, \nu) \leq \mathcal{W}_{p>1}(\mu, \nu)$ (Villani, 2008).

MDPs with Lipschitz Rewards. Given two metric spaces $(\mathcal{X}, d_\mathcal{X})$ and $(\mathcal{Y}, d_\mathcal{Y})$, a function $f : \mathcal{X} \to \mathcal{Y}$ is called 1-Lipschitz continuous if (Villani, 2008):

$$d_Y(f(x), f(x')) \leq d_X(x, x'), \forall (x, x') \in X \tag{3.4}$$

This implies that the Lipschitz semi-norm over the function space $\mathcal{F}(X, Y)$, defined as

$$\|f\|_L = \sup_{x \neq x'} \left\{ \frac{d_Y(f(x), f(x'))}{d_X(x, x')} \mid \forall (x, x') \in \mathcal{X} \right\} \tag{3.5}$$

is ≤ 1. When $(\mathcal{X}, d_\mathcal{X})$ is a Polish space and $\mu, \nu \in \mathcal{P}(\mathcal{X})$, the **Kantorovich-Rubinstein** formula states that (Villani, 2008):

$$\begin{aligned} \mathcal{W}_1(\mu, \nu) &= \sup_{\|f\|_L \leq 1} \left\{ \int_\mathcal{X} f \, d\mu - \int_\mathcal{X} f \, d\nu \right\} \\ &= \sup_{\|f\|_L \leq 1} \left\{ \mathbb{E}_\mu \left[f(X) \right] - \mathbb{E}_\nu \left[f(X) \right] \right\} \end{aligned} \tag{3.6}$$

where $\mathcal{W}_1(\mu, \nu)$ is the 1-Wasserstein distance between μ and ν with f as the cost function. Note that when $\|f\|_L \leq L_\mathcal{R}$ for any $L_\mathcal{R} > 0$, then the function f is called $L_\mathcal{R}$-Lipschitz continuous, and Equation 3.6 becomes (Gelada et al., 2019),

$$\mathcal{W}_1(\mu, \nu) = \frac{1}{L_\mathcal{R}} \sup_{\|f\|_L \leq L_\mathcal{R}} \left\{ \mathbb{E}_\mu \left[f(X) \right] - \mathbb{E}_\nu \left[f(X) \right] \right\}. \tag{3.7}$$

Now, we consider $\mathcal{X} = \mathcal{S} \times \mathcal{A}$, i.e. the state-action space, $\mathcal{Y} = \mathbb{R}$, i.e. the real line, and the function f to be the reward function $\bar{\mathcal{R}}$. Then, we can call the reward function $\bar{\mathcal{R}}$ to be $L_\mathcal{R}$-Lipschitz if

$$|\bar{\mathcal{R}}(s,a) - \bar{\mathcal{R}}(s',a')| \leq L_{\mathcal{R}} d_{\mathcal{SA}}((s,a),(s',a')) \tag{3.8}$$

for all $s, s' \in \mathcal{S}$, and $a, a' \in \mathcal{A}$, and $d_{\mathcal{SA}}((s,a),(s',a')) = d_{\mathcal{S}}((s,s')) + d_{\mathcal{A}}((a,a'))$ being the metric on the state-action space $\mathcal{S} \times \mathcal{A}$. If the reward function $\bar{\mathcal{R}}$ of an MDP is $L_{\mathcal{R}}$-Lipschitz, we refer it as an MDP with Lipschitz rewards. This approach follows the works of Pirotta et al. (2015) and Kallel et al. (2024); however this is a weaker condition than assuming a completely Lipschitz MDP. Pirotta et al. (2015) showed that for any pair of stationary policies π and π', the absolute difference between their corresponding objectives is

$$\left| J_\mu^\pi - J_\mu^{\pi'} \right| \leq \frac{L_{\mathcal{R}}}{\rho} \mathcal{W}_1(v_\pi, v_{\pi'}) \tag{3.9}$$

Proof. This stems from using Equation 3.2 to define the absolute performance difference of two policies π and π', with the initial state distribution $\mu(s)$ as,

$$\left| J_\mu^\pi - J_\mu^{\pi'} \right| = \frac{1}{\rho} \left| \mathbb{E}_{(s,a)\sim v_\pi} \left[\bar{\mathcal{R}}(s,a) \right] - \mathbb{E}_{(s,a)\sim v_{\pi'}} \left[\bar{\mathcal{R}}(s,a) \right] \right| \tag{3.10}$$

Then, the **Kantorovich-Rubinstein** formula dictates that (Gelada et al., 2019):

$$\sup_{\|\bar{R}\|_L \leq L_{\mathcal{R}}} \left| \mathbb{E}_{(s,a)\sim v_\pi} \left[\bar{\mathcal{R}}(s,a) \right] - \mathbb{E}_{(s,a)\sim v_{\pi'}} \left[\bar{\mathcal{R}}(s,a) \right] \right| = L_{\mathcal{R}} \mathcal{W}_1(v_\pi, v_{\pi'}) \tag{3.11}$$

By dividing both sides of Equation 3.11 by ρ, and omitting the supremum, the equation simplifies to Equation 3.9. □

3.4 RL Algorithms as Trajectories of Occupancy Measures

Exploration in RL generally refers to the problem of searching for an optimal policy that enables the agent to discover high-reward regions of the state-action space (Ibarz et al., 2021). Identifying the optimal policy is influenced by the *exploration-exploitation interplay* and the *learning strategy* (Kaelbling et al., 1996), which we collectively call the *exploration process*. The exploration process of an RL algorithm dictates how the policy model updates its policies (Sutton & Barto, 2018). During training, a *policy trajectory*, i.e. sequence of policies $(\pi_0, \pi_1, \ldots, \pi_N)$, is generated over policy updates due to the exploratory process. We assume these policies belong to a set of stationary Markov policies parameterized by $\theta \in \Theta$. For policies in this set $\pi_\theta \in \mathbf{\Gamma}_\Theta$, we define the space of occupancy measures corresponding to $\mathbf{\Gamma}_\Theta$ as $\mathcal{M} = \{v_{\pi_\theta}(s,a) \mid \pi_\theta \in \mathbf{\Gamma}_\Theta, \theta \in \Theta\}$.

Proposition 1 (Properties of $\mathcal{M}$)**.** *If the policy π has a smooth parameterisation θ, then the space of occupancy measures $\mathcal{M}$ is a differentiable manifold. (Proof in Appendix A.1)*

Note that a smoothly parameterised policy means that the $\theta \to \pi_\theta$ is C^∞, i.e. infinitely differentiable with respect to the parameters θ. A differentiable manifold (as established by Proposition 1) enables us to define smooth curves on the occupancy measure space. Thus, we endow the manifold $\mathcal{M}$ with a 1-Wasserstein metric $\mathcal{W}_1$ to the compute the length of any curve (path) on $\mathcal{M}$ since $(\mathcal{M}, \mathcal{W}_1)$ is a geodesic space (refer to Definition 1). The path distance between occupancy measures corresponding to policies parameterised by $\theta, \theta + d\theta \in \mathcal{M}$ is $ds = \mathcal{W}_1(v_{\pi_\theta}, v_{\pi_{\theta+d\theta}})$. Additionally, in imitation learning, the 1-Wasserstein distance between the occupancy measures of the learner and expert is often used as a minimisable loss function to learn the expert's policy (Zhang et al., 2020). Hence, the 1-Wasserstein distance reflects the effort required to achieve this imitation learning. Similarly, we propose the following quantification of the effort to update from one policy to another.

Definition 2 (Effort of Learning)**.** *We define the 1-Wasserstein metric between occupancy measures of two policies π and π', i.e. $\mathcal{W}_1(v_\pi, v_{\pi'})$, as the effort required to learn or update from one policy to the other.*

When a learning process engenders an update between occupancy measures in $\mathcal{M}$, we attribute the resulting update effort to the learning process and refer to it as the effort of learning. In a learning process, first the initial policy π_0 is obtained typically by randomly sampling the model parameters, then these parameters θ undergo updates until a predefined convergence criterion is satisfied, yielding the final optimal policy $\pi_N = \pi^*$. Since each policy has a corresponding occupancy measure, this process yields a sequence of points on $\mathcal{M}$, which can be connected by geodesics between successive points, producing a curve. The length of the curve is computed by the summation of the finite geodesic distances between consecutive policies along it (Lott, 2008),

$$C \triangleq \sum_{k=0}^{N-1} \mathcal{W}_1(v_{\pi_{\theta_k}}, v_{\pi_{\theta_{k+1}}}), \tag{3.12}$$

where θ_0 and θ_N are respectively the initial and final parameter values before and after learning.

3.4.1 Effort of Sequential Learning (ESL)

We established that RL generates a trajectory in the occupancy measure manifold $\mathcal{M}$, whose length is given by Equation 3.12. Compared to the long trajectory of sequential policies generated by the exploratory process, the geodesic L is the ideal shortest path to the optimal policy $\pi_N = \pi^*$ from π_0, whose length is $L = \mathcal{W}_1(v_{\pi_0}, v_{\pi_N})$. This path would be taken by an imitation-learning oracle algorithm that knows π^*. Both these paths are schematically depicted in Figure 3.1.

Definition 3 (Effort of Sequential Learning (ESL))**.** *We define the effort of sequential learning incurred by a trajectory of the exploratory process of an RL algorithm, relative*

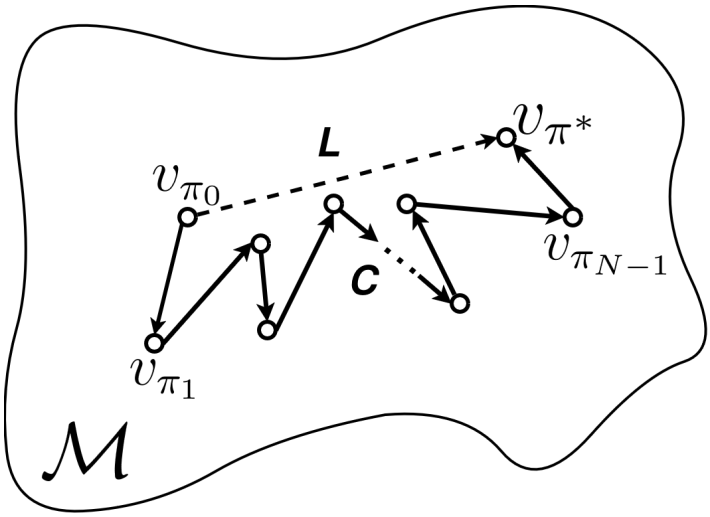


Figure 3.1: *Schematic of the policy trajectory C in the space of occupancy measures $\mathcal{M}$ during RL training (solid line) vs. the geodesic L (shortest path, dashed line) between the initial and final points (i.e. π_0 and $\pi_N = \pi^*$).*

to the oracle that knows $\pi^*(= \pi_N)$ as,

$$\eta \triangleq \frac{\sum_{k=0}^{N-1} \mathcal{W}_1(v_{\pi_k}, v_{\pi_{k+1}})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \tag{3.13}$$

Due to the stochasticity of the exploratory process, we introduce an expectation to obtain $\bar{\eta} = \mathbb{E}_{\pi_0,\mu}[\eta]$. We refer to $\bar{\eta}$ as the effort of sequential learning (ESL).

Note that $\bar{\eta} \geq 1$ and a larger $\bar{\eta}$ corresponds to a less efficient exploratory process of the RL algorithm. Hence, an RL algorithm with $\bar{\eta} \approx 1$ closely mimics the oracle and has an efficient exploratory process.

3.4.2 Optimal Movement Ratio (OMR)

Regret measures the total deviation in value functions incurred by a sequence of policies learned by an RL algorithm with respect to the optimal algorithm that always uses the optimal policy (Sinclair et al., 2023). We show that regret is connected to the sum of distances from each policy (in the sequence learned by an RL algorithm) to the optimal policy in the occupancy measure space.

Proposition 2 (Regret and Occupancy Measures)**.** *Given an MDP with $L_{\mathcal{R}}$-Lipschitz rewards, we obtain Regret $\triangleq \sum_{k=1}^{N} \left(J_\mu^{\pi^*} - J_\mu^{\pi_k}\right) \leq \frac{L_{\mathcal{R}}}{\rho} \sum_{k=1}^{N} \mathcal{W}_1(v_{\pi_k}, v_{\pi^*})$.*

Proof. Regret is determined by (Osband et al., 2013),

$$\text{Regret} = \mathbb{E}_{s\sim\mu}\left[\sum_k (V^*(s) - V_{\pi_k}(s))\right] \tag{3.14}$$

where $V^* = V_{\pi^*}$ is the value function of the optimal policy π^* while $V_{\pi_k}(s)$ is the value function of policy π_k, and μ is the initial state distribution. Since $J^\pi_\mu = \mathbb{E}_{s\sim\mu}[V_\pi(s)]$, we can use Equations 3.14 and 3.9 as follows,

$$\begin{aligned}
\text{Regret} &= \mathbb{E}_{s\sim\mu}\left[\sum_k (V^*(s) - V_{\pi_k}(s))\right] \\
&= \sum_k \left[\mathbb{E}_{s\sim\mu}(V^*(s) - V_{\pi_k}(s))\right] \\
&= \sum_k \left(J^{\pi^*}_\mu - J^{\pi_k}_\mu\right) \\
&= \sum_k \left|J^{\pi^*}_\mu - J^{\pi_k}_\mu\right| \\
&\leq \sum_k \frac{L_\mathcal{R}}{\rho}\mathcal{W}_1(v_{\pi^*}, v_{\pi_k})
\end{aligned} \tag{3.15}$$

□

We refer to $\mathcal{W}_1(v_{\pi_k}, v_{\pi^*})$ as the *distance-to-optimal*, and analogously use it as the expected immediate regret in the occupancy measure space. Furthermore, we refer to $\mathcal{W}_1(v_{\pi_k}, v_{\pi_{k+1}})$ as *stepwise-distance*. Interestingly, during training, the *distance-to-optimal* and *stepwise-distance* share a relationship illustrated in Figure 3.2.

Remark 1. *In Proposition 2 we notice that regret across policies measures the similarity of returns (with respect to the optimal policy), ignoring behavioural differences, like variations in actions at the same states. This makes regret advantageous in settings without critical safety and physical constraints, e.g. games, due to its computational efficiency. In contrast, the sum of distances-to-optimal focuses on behavioural differences between policies, is reward-agnostic, and is well-suited for environments where safety and physical constraints are critical, e.g. robotics. Whereas minimizing regret prioritizes matching the performance of the optimal policy, minimizing the sum of distances-to-optimal focuses on mimicking its behaviour. Thus, the sum of distances-to-optimal can be used similarly to regret, especially where the behaviour of achieving good performance is essential.*

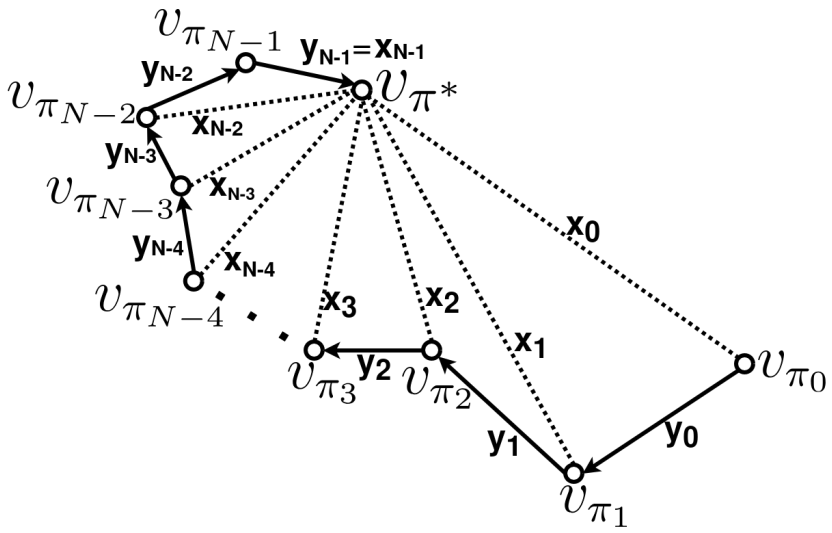


Figure 3.2: *Schematic of how distance-to-optimal (denoted by x_k) and stepwise-distance (denoted by y_k) on the occupancy measure space describe exploratory process of an RL algorithm during training.*

From Figure 3.2, we observe that if the change in *distance-to-optimal* is greater than zero, i.e. $\delta_k \triangleq \mathcal{W}_1(v_{\pi_k}, v_{\pi^*}) - \mathcal{W}_1(v_{\pi_{k+1}}, v_{\pi^*}) > 0$, then it indicates that the agent moved closer to the optimal policy. We define the set K^+ as containing indices k for which $\delta_k > 0$, while K^- contains the rest.

Definition 4 (Optimal Movement Ratio (OMR))**.** *We define the proportion of policy transitions that effectively reduce the distance-to-optimal, in a learning trajectory, as*

$$\kappa \triangleq \frac{\sum_{k \in K^+} \mathcal{W}_1(v_{\pi_k}, v_{\pi_{k+1}})}{\sum_{k=0}^{N-1} \mathcal{W}_1(v_{\pi_k}, v_{\pi_{k+1}})}\,. \tag{3.16}$$

Due to the stochasticity of the exploratory process, we introduce an expectation to obtain $\bar{\kappa} = \mathbb{E}_{\pi_0,\mu}[\kappa]$. We refer to $\bar{\kappa}$ as the optimal movement ratio (OMR).

Note that $\bar{\kappa} \in [0, 1]$, and $\bar{\kappa} \to 1$ indicates that nearly all the policy updates reduce the *distance-to-optimal*, thus showing high efficiency. $\bar{\kappa} \to 0$ implies low efficiency, since only a small fraction of the policy updates contributes toward the reduction of the *distance-to-optimal*. Additionally, we can observe how κ changes with respect to updates, by computing it from the i^{th} update onward till the end of the learning trajectory as,

$$\kappa(i) \triangleq \frac{\sum_{k \in K^+, k \geq i} \mathcal{W}_1(v_{\pi_k}, v_{\pi_{k+1}})}{\sum_{k=i}^{N-1} \mathcal{W}_1(v_{\pi_k}, v_{\pi_{k+1}})}, \text{ such that } i \in [0, N-T] \tag{3.17}$$

where $T \approx 0.1N$ to ensure that the last subsequence of policy updates have at least 10%

of the total updates in the trajectory. Similarly, $\bar{\kappa}(i) = \mathbb{E}_{\pi_0,\mu}[\kappa(i)]$. While $\bar{\kappa}$ considers the contribution of all policy transitions at once to reduce the *distance-to-optimal*, $\bar{\kappa}(i)$ dynamically tracks how the policy transitions contribute to reducing the *distance-to-optimal* as the RL agent evolves - allowing us to detect trends during training.

Remark 2. *The definitions of ESL and OMR assume that the policy at the end of learning is optimal. In Section 3.5.2, we define a version of ESL that is useful for the cases where an optimal policy is not reached. While this is not an empirical proxy, we show in Section 3.6.3 and Appendix B.1 that it is useful when the final policy is closer to optimal than the initial one. While regret also depends on an optimal policy, it is related to cumulative reward, whereas our metrics do not explicitly depend on rewards. Still, we showed a bound with regret in Proposition 2, and further discuss the possibility of extending our metrics to be reward-aware in Section 3.7. We show empirically that our metrics are complementary to regret in Section 3.6.2.*

3.4.3 Extension to Finite-Horizon Episodic Setting

In the episodic finite-horizon MDP formulation of RL, in short *Episodic RL* (Osband et al., 2013; Azar et al., 2017; Ouhamma et al., 2023), the agent interacts with the environment in multiple episodes of H steps. An episode starts by observing state s_1, then for $t = \{1, \dots, H\}$, the agent draws action a_t from a (possibly time-dependent) policy $\pi_t(\cdot \mid s_t)$, observes the reward $r(s_t, a_t)$, and transits to a state $s_{t+1} \sim T(\cdot \mid s_t, a_t)$. Here, the value function and the state-action value functions at step $h \in [H]$ are respectively defined as $V_h^\pi(s) \triangleq \mathbb{E}_\pi\left[\sum_{t=h}^H r(s_t, a_t) \mid s_h = s\right]$ and $Q_h^\pi(s,a) \triangleq \mathbb{E}_\pi\left[\sum_{t=h}^H r(s_t, a_t) \mid s_h = s, a_h = a\right]$. Following (Altman, 1999), we can define a finite-horizon version of occupancy measures as

$$v_\pi^H(s,a) \triangleq \frac{1}{H}\sum_{t=1}^{H} \mathbb{P}(s_t = s, a_t = a \mid \pi, \mu). \tag{3.18}$$

Following Syed et al. (2008), work by Kalagarla et al. (2021) shows that v_π^H can be used in the linear programming formulation for solving MDPs and satisfies the *Bellman Flow Constraints*. We prove that under some assumptions, the finite-horizon occupancy measures also construct a manifold, referred to as $\mathcal{M}^H$.

Proposition 3 (Properties of $\mathcal{M}^H$)**.** *If the policy π has a smooth parametrization θ, then the space of finite-horizon occupancy measures $\mathcal{M}^H$ is a differentiable manifold. (Proof in Appendix A.2)*

This allows us to similarly define a Wasserstein metric on this manifold, which in turn, allows us to compute ESL and OMR for evaluating different RL algorithms.

3.5 Computational Challenges and Solutions

Similar to regret, our method requires knowing the optimal policy. This is because the efficiency and effectiveness of exploratory processes of RL algorithms are highly coupled with their ability to reach the optimal policy. ESL and OMR depend on the policies being stationary and Markovian.

3.5.1 Policy datasets for computing occupancy measures

We consider approximations of occupancy measures using datasets assumed to be drawn from these measures. We estimate the Wasserstein distance between the occupancy measures using a method introduced by Alvarez-Melis & Fusi (2020) known as the *optimal transport dataset distance* (OTDD). OTDD uses datasets to estimate the Wasserstein distance between the underlying distributions as follows.

Suppose we have two datasets, each consisting of feature-label pairs, $\mathcal{D}_A = \{(t_A^i, u_A^i)\}_{i=1}^m \sim P_A(t,u)$ and $\mathcal{D}_B = \{(t_B^i, u_B^i)\}_{i=1}^n \sim P_B(t,u)$ with $t_A, t_B \in \mathcal{T}$ and $u_A, u_B \in \mathcal{U_A}, \mathcal{U_B}$. These datasets can be used to create empirical distributions $\hat{P}_A(t,u)$ and $\hat{P}_B(t,u)$. OTDD is the p-Wasserstein distance between the datasets $\mathcal{D}_A$ and $\mathcal{D}_B$ — which is essentially the distance between their empirical distributions $\hat{P}_A$ and $\hat{P}_B$ — with the cost function defined as the metric of the joint space $\mathcal{T} \times \mathcal{U}$ (Alvarez-Melis & Fusi, 2020). Naturally, the metric on this joint space can be defined as

$$d_{\mathcal{TU}}((t,u),(t',u')) = \left(d_{\mathcal{T}}(t,t')^p + d_{\mathcal{U}}(u,u')^p\right)^{1/p}, \quad \text{for} \quad p \geq 1 \tag{3.19}$$

However, in most applications $d_{\mathcal{T}}$ is readily available, while $d_{\mathcal{U}}$ might be scarce, especially in Supervised Learning (SL) settings between labels from unrelated label sets (Alvarez-Melis & Fusi, 2020). Additionally, we want $d_{\mathcal{T}}$ and $d_{\mathcal{U}}$ to have the same units to be summable. To overcome these issues, $d_{\mathcal{U}}$ is expressed in terms of $d_{\mathcal{T}}$ by mapping labels u to distributions over the feature space $\mathcal{P}(\mathcal{T})$ as $u \to \alpha_u(T) \triangleq P(T \mid U = u) \in \mathcal{P}(\mathcal{T})$ (see Appendix B.3 for a discussion about the limitations of determining such a mapping in practice). Therefore, the distance between the labels u and u' is defined as the p-Wasserstein distance between $\alpha_u(T)$ and $\alpha_{u'}(T)$,

$$\begin{aligned} d_{\mathcal{U}}(u,u')^p &= \mathcal{W}_p^p(\alpha_u(T), \alpha_{u'}(T)) \\ &= \min_{\pi \in \Pi(\alpha_u, \alpha_{u'})} \int_{\mathcal{T}\times\mathcal{T}} (d_{\mathcal{T}}(t,t'))^p \, d\pi(t,t') \end{aligned} \tag{3.20}$$

The metric on the joint space becomes,

$$d_{\mathcal{TU}}((t,u),(t',u')) = \left(d_{\mathcal{T}}(t,t')^p + \mathcal{W}_p^p(\alpha_u(T), \alpha_{u'}(T))\right)^{1/p} \tag{3.21}$$

If we let $\mathcal{Z} = \mathcal{T} \times \mathcal{U}$, then the p-Wasserstein distance between $\hat{P}_A(t, u)$ and $\hat{P}_B(t, u)$ is a *nested* Wasserstein distance:

$$\begin{aligned} \mathcal{W}_p^p(\hat{P}_A, \hat{P}_B) &= \min_{\pi \in \Pi(P_A, P_B)} \int_{\mathcal{Z} \times \mathcal{Z}} (d_{\mathcal{Z}}(z, z'))^p \, d\pi \\ &= \min_{\pi \in \Pi(P_A, P_B)} \int_{\mathcal{TU} \times \mathcal{TU}} \left(d_{\mathcal{T}}(t, t')^p + \mathcal{W}_p^p(\alpha_u, \alpha_{u'}) \right) \, d\pi \end{aligned} \tag{3.22}$$

$\mathcal{W}_p^p(\hat{P}_A, \hat{P}_B)$ is the OTDD between datasets $\mathcal{D}_A$ and $\mathcal{D}_B$, often expressed as $d_{OT}(\mathcal{D}_A, \mathcal{D}_B)$, i.e. $d_{OT}(\mathcal{D}_A, \mathcal{D}_B) = \mathcal{W}_p^p(\hat{P}_A, \hat{P}_B) \approx \mathcal{W}_p^p(P_A, P_B)$. This is used in transfer learning to determine the distance (or similarity) between datasets. Figure 3.3 illustrates OTDD when applied to RL using datasets of state-action pairs. With the tool to estimate the Wasserstein distance between datasets established, we introduce *policy datasets*.

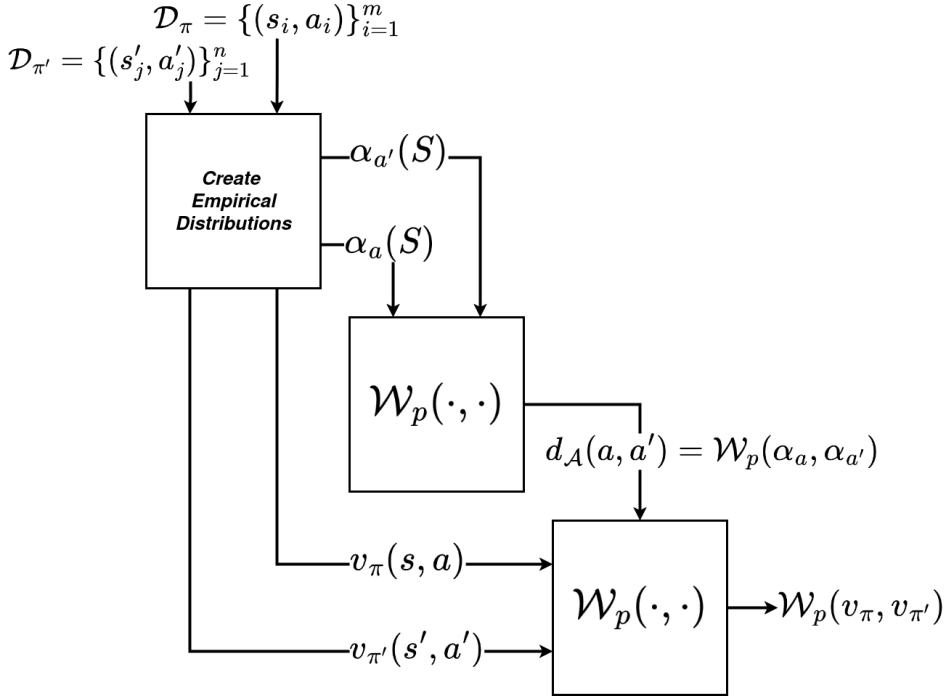


Figure 3.3: *Illustration of OTDD when applied to RL.*

Definition 4 (Policy dataset). *A dataset of a policy $\mathcal{D}_\pi$ is a set of state-action pairs drawn from the policy's occupancy measure, i.e. $\mathcal{D}_\pi = \{(s_{(i)}, a_{(i)})\}_{i=1}^m \sim v_\pi(s, a)$. These can be constituted from the rollouts generated by the policy during task execution.*

We know from imitation learning that if we are given $\mathcal{D}_\pi$, generated by an expert policy, we can train a policy model on it in a supervised manner via behaviour cloning (Hussein et al., 2017). Thus, knowing $\mathcal{D}_\pi$ can allow converting an RL task into a SL task. Consider a scenario when we have access to a sequence of datasets $(\mathcal{D}_{\pi_0}, \ldots, \mathcal{D}_{\pi_N})$, each corresponding to policy π_k for $k \geq 0$. If we train (in a supervised manner) a policy

model sequentially on these datasets, the model will undergo a similar policy evolution as the RL algorithm that generated the policy trajectory $(\pi_k)_{k\geq 0}$. This allows us to conceptualise learning in RL as a sequence of SL tasks with sequential transfer learning across the datasets $(\mathcal{D}_{\pi_0}, \ldots, \mathcal{D}_{\pi_N})$, as illustrated in Figure 3.4. We employ OTDD to estimate $\mathcal{W}_1(v_{\pi_k}, v_{\pi_{k+1}})$ using these datasets, i.e. $d_{OT}(\mathcal{D}_{\pi_k}, \mathcal{D}_{\pi_{k+1}}) \approx \mathcal{W}_1(v_{\pi_k}, v_{\pi_{k+1}})$, based on Proposition 4.

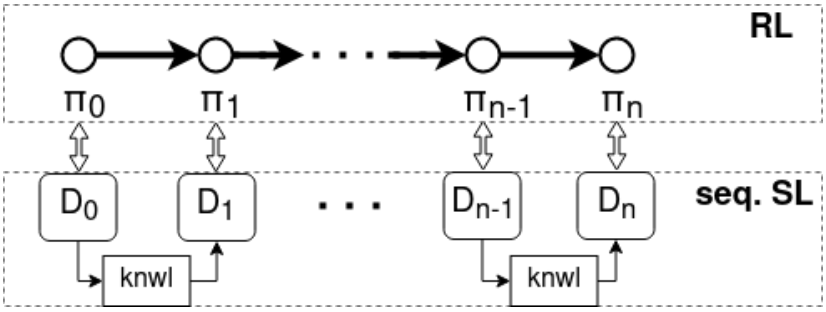


Figure 3.4: *Illustration of **RL** as a sequence of SL tasks (**seq. SL**). Policies π_n encountered during learning in RL have corresponding datasets $\mathcal{D}_n$ of state-action pairs. Training a classifier successively on these datasets, while transferring knowledge (**knwl**), is analogous to training the RL agent.*

Proposition 4 (Upper Bound on Estimation Error)**.** *Let an RL algorithm yield a sequence of policies $\pi_0, \ldots, \pi_N$ while training. Now, we construct N datasets $\mathcal{D}_{\pi_0}, \ldots, \mathcal{D}_{\pi_N}$, each consisting of M rollouts of the corresponding policies. Then, we can use these datasets to approximate $\sum_{k=0}^{N-1} \mathcal{W}_1(v_{\pi_{\pi_k}}, v_{\pi_{\pi_{k+1}}})$ by $\sum_{k=0}^{N-1} d_{OT}(\mathcal{D}_{\pi_k}, \mathcal{D}_{\pi_{k+1}})$ with an expected error upper bound $\frac{2N\mathcal{E}_2}{\sqrt{M}} + N\gamma^{T+1} diam(\mathcal{SA})$. Here, T is the total number of steps per episode, $diam(\mathcal{SA})$ is the diameter of the state-action space, and $\mathcal{E}_2$ is a positive-valued and polylogarithmic function of $\mathcal{S}$ and $\mathcal{A}$. For finite horizon case, we can further reduce the error bound to $\frac{2N\mathcal{E}_2}{\sqrt{M}}$.*

Proof of proposition 4 is provided in Appendix A.3. The results support that ESL and OMR can be estimated as,

$$\bar{\eta} = \mathbb{E}_{\pi_0,\mu}\left[\frac{\sum_{k=0}^{N-1} d_{OT}(\mathcal{D}_{\pi_k}, \mathcal{D}_{\pi_{k+1}})}{d_{OT}(\mathcal{D}_{\pi_0}, \mathcal{D}_{\pi_N})}\right] \tag{3.23}$$

$$\bar{\kappa} = \mathbb{E}_{\pi_0,\mu}\left[\frac{\sum_{k\in K^+} d_{OT}(\mathcal{D}_{\pi_k}, \mathcal{D}_{\pi_{k+1}})}{\sum_{k=0}^{N-1} d_{OT}(\mathcal{D}_{\pi_k}, \mathcal{D}_{\pi_{k+1}})}\right] \tag{3.24}$$

3.5.2 When an optimal policy is not reached

So far we have assumed that the algorithms converge at the optimal policy, i.e. $\pi_N = \pi^*$. However, this is not always true. We consider a scenario when $\pi_N \neq \pi^*$, and define the *suboptimal Effort of Sequential Learning* (ESL-sub) as,

$$\eta_{sub} = \frac{\sum_{k=0}^{N-1} \mathcal{W}_1(v_{\pi_{\pi_k}}, v_{\pi_{\pi_{k+1}}})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})}, \quad \pi_N \neq \pi^* \tag{3.25}$$

The relationship between ESL and ESL-sub is described below in Proposition 5 as follows:

Proposition 5. *Given $\pi_0 \neq \pi_N \neq \pi^*$, we obtain*

$$\frac{\eta_{sub} - \eta}{\eta + 1} \leq \frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \tag{3.26}$$

Proof. By definition of η_{sub} (Equation 3.25), we get

$$\begin{aligned}
\eta_{sub} &= \frac{\sum_{i=0}^{N-2} \mathcal{W}_1(v_{\pi_i}, v_{\pi_{i+1}}) + \mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi_N})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \\
&= \frac{\sum_{i=0}^{N-2} \mathcal{W}_1(v_{\pi_i}, v_{\pi_{i+1}}) + \mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi_N})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \times \frac{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})} \\
&= \frac{\sum_{i=0}^{N-2} \mathcal{W}_1(v_{\pi_i}, v_{\pi_{i+1}})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})} \times \frac{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} + \frac{\mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi_N})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})}
\end{aligned} \tag{3.27}$$

By definition of η (Equation 3.13), we get

$$\begin{aligned}
\eta &= \frac{\sum_{i=0}^{N-2} \mathcal{W}_1(v_{\pi_i}, v_{\pi_{i+1}}) + \mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})} \\
&= \frac{\sum_{i=0}^{N-2} \mathcal{W}_1(v_{\pi_i}, v_{\pi_{i+1}})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})} + \frac{\mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})}
\end{aligned} \tag{3.28}$$

By rearranging Equation 3.28,

$$\frac{\sum_{i=0}^{N-2} \mathcal{W}_1(v_{\pi_i}, v_{\pi_{i+1}})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})} = \eta - \frac{\mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})} \tag{3.29}$$

Substitute Equation 3.29 into Equation 3.27,

$$\begin{aligned}
\eta_{sub} &= \left(\eta - \frac{\mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})}\right) \times \frac{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} + \frac{\mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi_N})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \\
&= \eta \frac{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} - \frac{\mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} + \frac{\mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi_N})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \\
&= \eta \frac{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} + \frac{\mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi_N}) - \mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \\
&\leq \eta \frac{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} + \frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})}
\end{aligned} \tag{3.30}$$

The inequality above is true due to the reverse triangle inequality $\mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi_N}) - \mathcal{W}_1(v_{\pi_{N-1}}, v_{\pi^*}) \leq \mathcal{W}_1(v_{\pi_N}, v_{\pi^*})$. By applying another triangle inequality, we get

$$\mathcal{W}_1(v_{\pi_0}, v_{\pi^*}) \leq \mathcal{W}_1(v_{\pi_0}, v_{\pi_N}) + \mathcal{W}_1(v_{\pi_N}, v_{\pi^*})$$

This implies that

$$\frac{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \leq \frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} + 1 \tag{3.31}$$

By multiplying both sides of Equation 3.31 by η and adding $\frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})}$ to them, we get

$$\eta \frac{\mathcal{W}_1(v_{\pi_0}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} + \frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \leq \eta \left(\frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} + 1 \right) + \frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \tag{3.32}$$

The left-hand side of Equation 3.32 appears on the right-hand side of Equation 3.30, therefore

$$\begin{aligned} \eta_{sub} &\leq \eta \left(\frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} + 1 \right) + \frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \\ &= \eta + \eta \frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} + \frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \\ &= \eta + \frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} (\eta + 1) \end{aligned} \tag{3.33}$$

which simplifies to

$$\begin{aligned} \eta_{sub} - \eta &\leq \frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} (\eta + 1) \\ \frac{\eta_{sub} - \eta}{\eta + 1} &\leq \frac{\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})}{\mathcal{W}_1(v_{\pi_0}, v_{\pi_N})} \end{aligned}$$

□

Remark 3. *Note that Equation 3.26 shows that when π_N is close to π^*, then η_{sub} is a good approximation of η, and thus a good quantifier to determine the efficiency of the algorithm's exploratory process. However, $\mathcal{W}_1(v_{\pi_N}, v_{\pi^*})$ is dependent on the RL algorithm and hence a bound cannot be provided here. Still, η_{sub} might be useful when π_N is closer to π^* than π_0. A fallible proxy for this could be when the performance of π_N is better than of π_0, i.e. $J_\mu^{\pi_N} > J_\mu^{\pi_0}$. We show the usefulness of η_{sub} in our experimental results in Section 3.6.3 and Appendix B.1 for simple environments. It remains to be seen how useful η_{sub} is in complex environments.*

3.5.3 Application in Episodic Settings

In Section 3.4.3, we introduced the finite-horizon occupancy measure as,

$$v_\pi^H(s,a) = \frac{1}{H}\sum_{t=1}^{H} \mathbb{P}(s_t = s, a_t = a \mid \pi, \mu)$$

for which the probability of the state-action pair selected is time-dependent because the policy is time-dependent $\pi_t(a_t|s_t)$. The equation is general for finite-horizon settings, however in practice, especially for episodic RL, analysis is restricted to stationary policies (Memmel et al., 2022; Aleksandrowicz & Jaworek-Korjakowska, 2023; Liu, 2023). This consequently makes the probability of the state-action pair time-independent and thus simplifies to,

$$v_\pi^H(s,a) = \mathbb{P}(s, a \mid \pi, \mu) \tag{3.34}$$

The reason is that the policy selects the same action a in state s regardless of time. Therefore stationary occupancy measures are induced, where the expected number of state-action pair visits are independent of the time-step. Bojun (2020) provides extensive details about the existence of stationarity in episodic MDPs and shows (in Theorem 4) that

$$\mathbb{E}_{(s,a)\sim v_\pi^H}\left[\bar{\mathcal{R}}(s,a)\right] = \frac{\mathbb{E}_{\zeta\sim M_\pi}\left[\sum_{t=1}^{H(\zeta)} R(s_t,a_t)\right]}{\mathbb{E}_{\zeta\sim M_\pi}\left[H(\zeta)\right]} \tag{3.35}$$

where ζ is the episodic state-action pair trajectory, $H(\zeta)$ is the episode length corresponding to ζ, and M_π is the markov chain induced by policy π. The correctness of our v_π^H computation can be verified by calculating the relative error derived from Equation 3.35 to check its validity. The relative error is defined as,

$$\text{Rel. Error } \% \;= 100 \times \frac{\mathbb{E}_{(s,a)\sim v_\pi^H}\left[\bar{\mathcal{R}}(s,a)\right]\mathbb{E}_{\zeta\sim M_\pi}\left[H(\zeta)\right] - \mathbb{E}_{\zeta\sim M_\pi}\left[\sum_{t=1}^{H(\zeta)} R(s_t,a_t)\right]}{\mathbb{E}_{\zeta\sim M_\pi}\left[\sum_{t=1}^{H(\zeta)} R(s_t,a_t)\right]} \tag{3.36}$$

The validation results of our method to compute v_π^H are presented in Appendix B.2.

Estimation of ESL and OMR. We use Algorithm 1 to determine ESL and OMR, where the policy datasets are collected for every policy model update until convergence. Note that for brevity we represent π_{θ_k} by θ_k and assume $\theta_K = \theta^*$.

Algorithm 1: Estimating ESL and OMR

Input: MDP, Algorithm
Output: ESL, OMR

Initialise model parameters $\theta \sim p(\theta)$ and set variables $C = 0$, $J = 0$;
Perform standard RL training;
for $k = 0, 1, \ldots, K$ *model updates* **do**
    for $t = 0, 1, \ldots, N$ *episodes* **do**
        Evaluate policy θ_k;
        Store $\{(s_{(i)}, a_{(i)})\}_{i=0}^{H(t)}$ in $\mathcal{D}_k$; /*$H(t)$ is length of episode t*/
$L \leftarrow d_{OT}(\mathcal{D}_0, \mathcal{D}_K)$;
for $k = 0, 1, \ldots, K-1$ **do**
    $C \leftarrow C + d_{OT}(\mathcal{D}_k, \mathcal{D}_{k+1})$;
    if $d_{OT}(\mathcal{D}_K, \mathcal{D}_{k+1}) - d_{OT}(\mathcal{D}_K, \mathcal{D}_k) > 0$ **then**
        $J \leftarrow J + d_{OT}(\mathcal{D}_k, \mathcal{D}_{k+1})$;
ESL $\leftarrow C/L$, OMR $\leftarrow J/C$;

3.6 Experimental Evaluation

In this section, we evaluate the proposed methods in the *2D-Gridworld* and *Mountain Car* (Moore, 1990; Brockman et al., 2016) environments, to analyse our methods in discrete and continuous state-action spaces respectively. Note that the details of hyperparameter settings of the algorithms used in this section are provided in Appendix B.5.1. Our code is available at: `https://github.com/nkhumise-rea/analysis_of_occupancy_measure_trajectory`.

2D-Gridworld. The environment is of size 5×5 with actions: {up, right, down, left}. The start and goal states are always located at top-left $(0, 0)$ and bottom-right $(4, 4)$ respectively. In the gridworld, we perform experiments on three settings namely:- A) deterministic with dense rewards, B) deterministic with sparse rewards, and C) stochastic with dense rewards. In A) *Deterministic, dense rewards setting*: State transitions are deterministic. The reward function is given by $\|s_t - s_g\|_1$, where s_t is the agent state at timestep t and s_g is the goal state.

In B) *Deterministic, sparse rewards setting*: State transitions are deterministic and all states issue a reward of -0.04 except the goal state with reward of +1. In C) *Stochastic, dense rewards setting*: Actions have a probability of 0.8 in the instructed direction and 0.1 in each adjacent direction. The reward function is the same as in setting A. Note that we used $L1$ distance as metric ($d_{\mathcal{X}}$) for the state space of this environment which underpins the $\mathcal{W}_1$.

Mountain Car. The environment, in our experimentation, is a deterministic MDP with dense rewards that consists of both continuous states and actions. The environment introduced by Moore (1990), is a 2-dimensional model of an underpowered vehicle

located between two hills, one behind and the other in front of the vehicle. The vehicle must exploit its momentum to ascend the front hill and reach a goal state at the top. Due to inadequate engine power, the vehicle must first move away from the goal (i.e. up the hill behind) to build sufficient momentum before climbing the front hill. The reward function employed issues a reward of -1 at every timestep in which the agent has not reached the goal (i.e. $s_t \neq s_g$) and 0 when the goal is reached. Note that we utilised the Gymnasium API (Brockman et al., 2016) to conduct experiments on this environment. Furthermore, the $L2$ distance served as a metric ($d_{\mathcal{X}}$) for the state space.

Our experiments aim to address the following questions:
1. *What information can the visualization of the policy evolution during RL training provide about the exploratory process of the algorithm?*
2. *How do ESL and OMR allow us to analyze the exploratory processes of RL algorithms?*
3. *Does ESL scale proportionally with task difficulty?*

Summary of Results. In Section 3.6.1, we demonstrate that visualizing the evolution of *distance-to-optimal* and *stepwise-distance* of different RL algorithms during training reveals: 1) whether the agent is stuck in suboptimal policies, 2) the coverage area of the exploration processes, and 3) their varied characteristics over time. We further compare ESL and OMR of different algorithms on a few environments in Section 3.6.2. Finally, we show in Section 3.6.4 that ESL scales proportionally with task difficulty, and thus, reflects the effects of task difficulty on exploration and learning.

3.6.1 Exploration Trajectories of RL Algorithms

(I) Discrete MDP. To understand the utility of visualizing exploratory processes, we use the following RL algorithms: 1) Tabular Q-learning (Watkins & Dayan, 1992) with ϵ-greedy a) $\epsilon = 0$ and b) $\epsilon = 1$ strategies; 2) UCRL2 (Jaksch et al., 2010); 3) PSRL (Osband et al., 2013); 4) SAC (Haarnoja et al., 2018; Christodoulou, 2019); and 5) DQN (Mnih et al., 2013) with a decaying ϵ-greedy exploration strategy. The algorithms solve the gridworld under *Setting A* (deterministic with dense rewards). Figure 3.5 presents exploratory behaviours of the algorithms in both the occupancy measure space and state space.

Q-learning: $\epsilon = 0$ vs $\epsilon = 1$. Note that $\epsilon = 0$ updates the Q-table by only exploiting, while $\epsilon = 1$ by exploring. From the state visitations, we observe expected characteristics such as (a) a preferred visitation path for $\epsilon = 0$ and (b) for $\epsilon = 1$, the visitation frequencies are similar at states equidistant from the start-state. Additionally, for $\epsilon = 1$, the visitation frequencies gradually decrease as the distance from the start-state increases. From the policy evolution, we see how scattered and erratic the policy transitions are for $\epsilon = 0$. But $\epsilon = 1$ is characterised by extended periods during which the policy experiences parameter updates however exhibits negligible change in its *distance-to-optimal* while its *stepwise-distance* is nearly zero (see Figure B.7 for better illustration). In Figure 3.5, this behaviour appears as vertical line segments (parallel

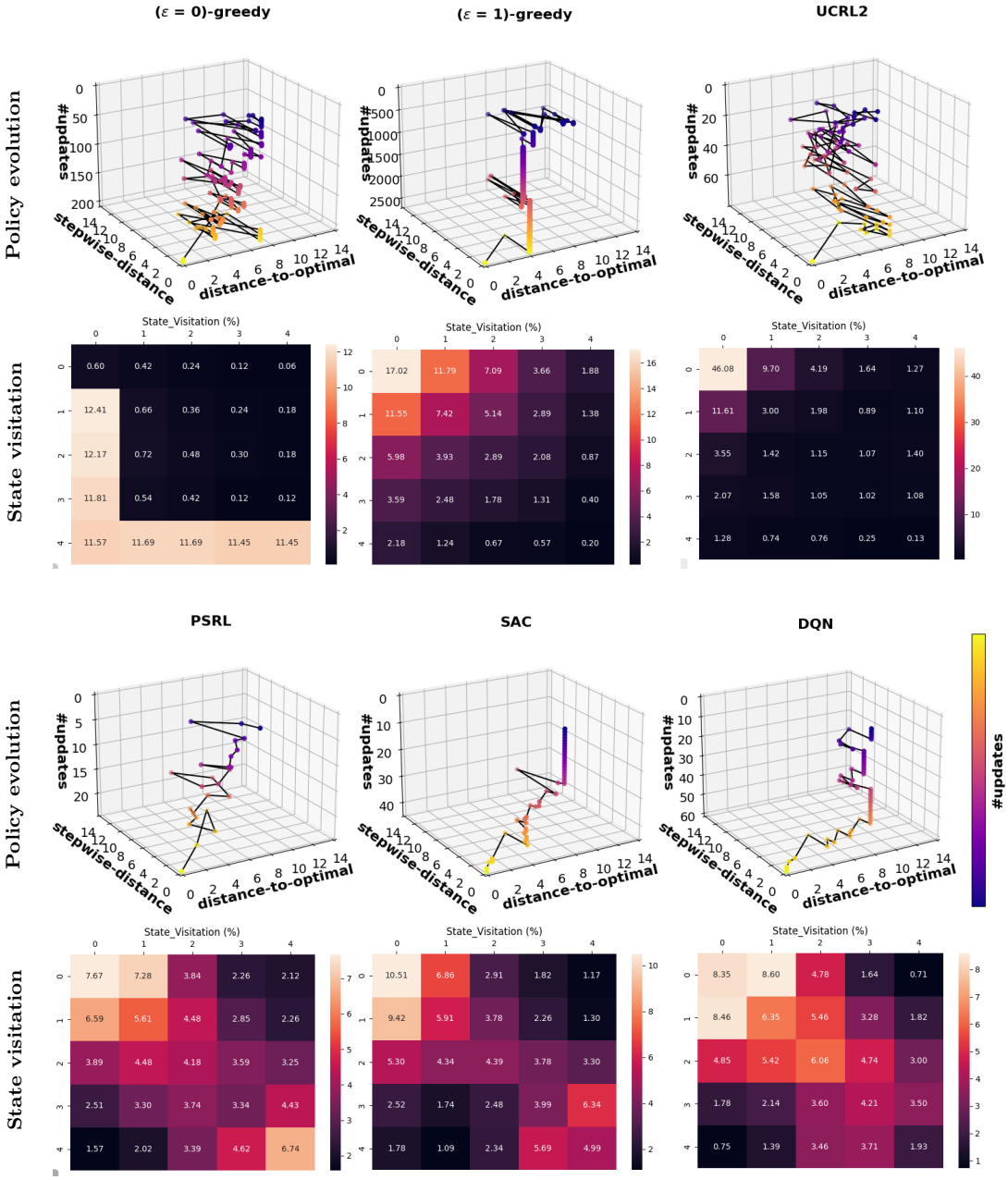


Figure 3.5: 1^{st} *and* 3^{rd} *rows: 3D plots of distance-to-optimal (x-axis) and stepwise-distance (y-axis) across number of updates (z-axis), illustrating policy evolution in the occupancy measure space for algorithms:* $\epsilon(=0)$*-greedy and* $\epsilon(=1)$*-greedy Q-learning, UCRL2 (*1^{st} *row, left to right); and PSRL, SAC, and DQN (*3^{rd} *row, left to right).* 2^{nd} *and* 4^{th} *rows: Corresponding state visitation frequencies over the full training. The problem setting is deterministic with dense rewards and* 15 *maximum number of steps per episode. (NB. 2D projections of these policy evolution plots are in Figure B.7 at Appendix B.4.3, and corresponding performance plots are in Figure B.5a at Appendix B.4.2.)*

to the *#updates* axis), denoting that successive updates do not produce measurable progress toward the optimal policy and the algorithm transitions between policies in close proximity. This pattern exhibits stagnation within a suboptimal region of the policy space.

In this setting, we observe that $\epsilon = 0$ is characterised by transitioning between diverse policies (i.e. having a larger coverage area) while $\epsilon = 1$ is likely to remain in the suboptimal region. This stagnation arises from the high degree of action randomness under $\epsilon = 1$, which causes the agent to select suboptimal actions. This yields limited information gain, and thus slows convergence of the Q-table, delaying policy improvement until higher-value actions are discovered.

UCRL2 vs PSRL. UCRL2 has nearly uniform state visits (with the exception of the start-state because the initial state distribution is 1 at state $(0, 0)$), thus being consistent with literature since the algorithm selects exploratory state-action pairs more uniformly (Jaksch et al., 2010). In contrast, PSRL has high visit frequencies along the diagonal states, because it selects actions according to the probability that they are optimal (Osband et al., 2013). We observe from the policy evolution plots that PSRL has smoother policy transitions that are orientated towards optimality, while UCRL2 behaves more aggressively with policy transitions that do not taper as it approaches optimality. Osband et al. (2013) highlighted that exploration in PSRL is guided by the variance of sampled policies as opposed to optimism in UCRL2. We observe in Figure 3.5 that the guiding variance in PSRL reduces after every policy update until the optimal policy is reached, while UCRL2 maintains high variance.

SAC vs DQN. The state visits of both the algorithms appear to be similar. SAC has higher visitation frequencies at the corners than DQN. Surprisingly from the policy evolution plots, we learn that both algorithms have a reluctance to transition between diverse policies — hence they exhibit vertical line segments in Figure 3.5 that denote stagnation within a suboptimal region of the policy space (as previously discussed). This reluctance is due to the slow 'soft updates' of target networks (Lillicrap et al., 2016) in the algorithms. These 'soft updates' refer to updating the target networks periodically instead of every step (Mnih et al., 2015), as performed by the online Q-function networks. We also observe that SAC approaches optimality more gradually than DQN.

All algorithms. Figure 3.5 shows that Q-learning ($\epsilon = 0$)-greedy and UCRL2 were more meandering (with larger coverage area) towards optimality than the rest. Although the algorithms are fundamentally different, the visualizations of their policy evolutions are quite similar. SAC and DQN approach the optimal policy more directly and smoothly (with smaller coverage area) than the rest. These characteristics are intuitively revealed by the *policy evolution* plots, are aligned with literature, and hence enhance our understanding of the exploratory processes.

(II) Continuous MDP. We use DDPG (Lillicrap et al., 2016) and SAC to solve the Mountain Car problem. The policy evolutions of these algorithms are presented in

Figure 3.6.

DDPG vs SAC. Both exhibit short-distances (< 1) between policy updates (i.e. small coverage area). They depict no sign of being stuck or settling early on any particular policy, which shows their continuously exploratory nature. While they begin with almost constant mean *distances-to-optimal* and *stepwise-distances*, SAC drops its mean *distance-to-optimal* earlier than DDPG.

Figure 3.6 illustrates how OMR changes with update number k (refer to Equation 3.17). For both algorithms, OMR(k) remains near chance level (~ 0.5) initially, then sharply increases near the final updates. This suggests that early policy updates are purely exploratory and oblivious to policy improvement but align with the optimal policy just before convergence. The efficiency of the algorithm depends on how early this transition occurs, e.g. starts earlier for SAC than DDPG, rendering SAC more efficient.

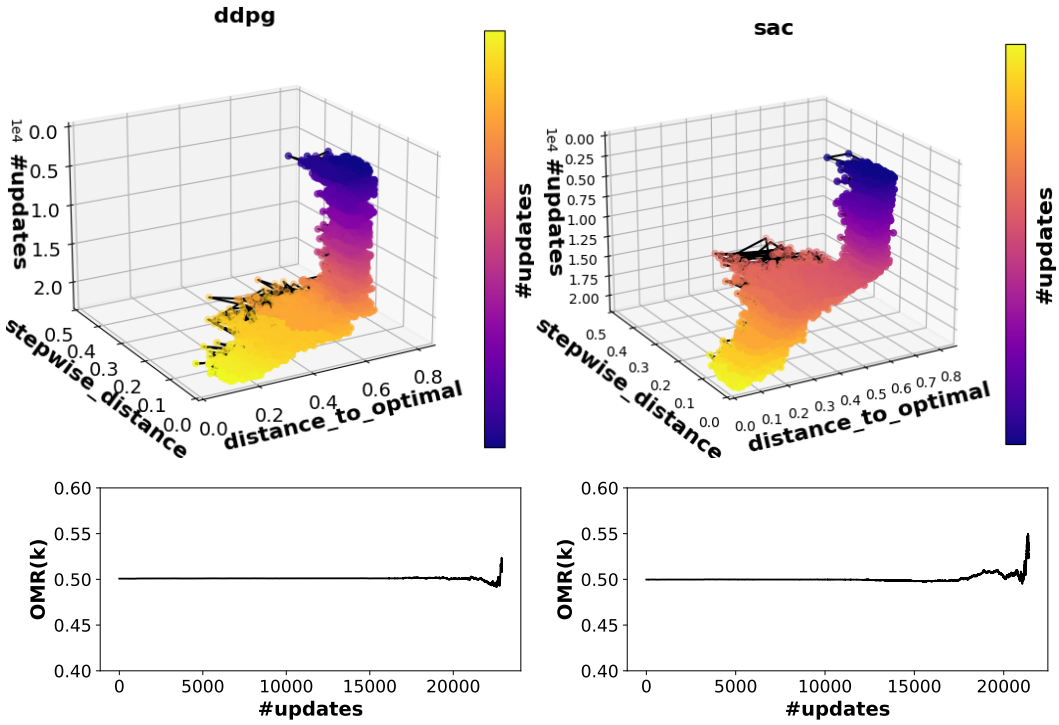


Figure 3.6: *Top row: 3D scatter plots of distance-to-optimal and stepwise-distance vs. number of updates for DDPG and SAC. Bottom row: OMR(k) vs. #update k for the corresponding algorithms. Note that corresponding performance plots are in Figure B.5b in Appendix B.4.2*

3.6.2 Comparison of ESL and OMR across RL Algorithms and Environments, and their complementarity to number of updates and regret

Tables 3.1—3.3 showcase how ESL and OMR are summary indices of the policy trajectories during learning by evaluating the algorithms in various settings.

Table 3.1: *Evaluation of RL algorithms (over 40 runs) in the **deterministic, dense-rewards setting** for the gridworld, including Effort of Sequential Learning (ESL), Optimal Movement Ratio (OMR), number of updates to convergence (UC), and success rate (SR). Lowest ESL, lowest UC, and highest OMR values are in **bold**. The highest ESL value is starred ($\star$).*

Algo.	ESL	OMR	UC	SR%
SAC	9.26±5.54	0.58±0.14	980±670	100
UCRL2	47.2±8.20*	0.49±0.04	60.7±11	100
PSRL	23.2±11.5	0.52±0.06	**34.1±9.34**	100
DQN	12.4±7.13	0.54±0.11	161±93	98
ϵ(=1)-greedy	**6.27±2.22**	**0.61±0.09**	672±385	100
ϵ(=0.9)-decay	8.10±3.43	0.61±0.10	389±138	100
ϵ(=0)-greedy	15.5±5.28	0.53±0.06	176±37.9	84

Table 3.2: *Evaluation of RL algorithms (over 40 runs) in the **deterministic, sparse-rewards** and **stochastic, dense-rewards** settings for the gridworld. Lowest ESL, highest OMR and lowest UC values are in **bold**. The highest ESLs are starred.*

Algo.	ESL	OMR	UC	SR%
Deterministic, sparse				
SAC	**27.8±21.9**	**0.57±0.13**	4385±3274	100
UCRL2	73.3±0.0	0.45±0.0	**93.0±0.0**	100
PSRL	73.2±54.1	0.52±0.076	100±67.3	100
DQN	137±154*	0.49±0.08	12638±4431	80
Stochastic, dense				
SAC	445±245	0.501±0.004	2463±2043	**92**
UCRL2	198±121	0.502±0.027	268±155	32
PSRL	**55.4±33.6**	**0.52±0.04**	**76.1±50.6**	**92**
DQN	458±311*	0.502±0.01	1586±1077	24

Dense Rewards. We observe, in Table 3.1, that PSRL took the lowest number of updates (UC) to reach the optimal policy in contrast with SAC. Yet, PSRL was meandering more than SAC (see Figure 3.5). The relative directness of SAC is captured by lower ESL and higher OMR compared to PSRL. Even though SAC has larger UC than PSRL, it took a shorter path to optimality than PRSL. This shows that UC does not necessary correlate with ESL and OMR, and it provides incomplete information about the exploratory processes. Indeed, two algorithms may have the same UC, but different ESL and/or OMR due to different *step-wise distances* and varied movement towards optimality.

Sparse Rewards. In the sparse rewards setting (Table 3.2), low performance of DQN is observed in both our indices and UC. However, SAC is more efficient with lowest ESL and highest OMR, yet UCRL2 has the lowest number of updates (UC). Note that

UCRL2 is provably regret-optimal, while SAC does not have such rigorous theoretical guarantees but is known to be practically efficient, and this is well captured by ESL and OMR. So far, Tables 3.1 and 3.2 demonstrate how our indices provide a clearer picture of exploratory processes than the number of updates.

Stochastic Transitions. In the stochastic setting (Table 3.2), by observing only successful cases, we notice that the meandering characteristic of PSRL and UCRL2 is more suitable for this setting than SAC and DQN (based on better ESL and OMR values). PSRL and UCRL2 have similar regret bounds (Osband et al., 2013); yet in Tables 3.1 and 3.2, PSRL has better ESL and OMR (along with a higher success rate). This aligns with the regret analysis presented in (Osband et al., 2013), which shows that PSRL achieves lower regret than UCRL2 in certain environments.

Table 3.3 corroborates with the policy evolution plots in Figure 3.6, in that due to SAC dropping its mean *distance-to-optimal* earlier than DDPG it exhibits a lower ESL. Additionally, we notice a trend of increasing ESL and decreasing OMR across algorithms when shifting from dense-rewards to sparse-rewards settings, from deterministic to stochastic transitions, from discrete to continuous environments, indicating an increase in the effort of the exploratory processes. We have shown that ESL and OMR enhance the understanding of exploratory processes by effectively summarizing the policy trajectories of algorithms during learning. They offer more insight than the number of updates, and align with regret when algorithms reach an optimal policy. In the next Section, we demonstrate the utility of ESL when an optimal policy is not reached.

Table 3.3: *Evaluation of RL algorithms in the Mountain Car continuous MDP (over* 5 *runs). The variances for OMR are negligible.*

Algo.	ESL	OMR	UC	SR%
DDPG	1881±500	0.501	23500±5268	100
SAC	1619±189	0.5	22700±2971	100

3.6.3 Usefulness of ESL when optimal policy is not reached

When an optimal policy is not reached at the end of learning, ESL cannot be computed exactly. At this point, we propose to use an approximation of ESL, i.e. η_{sub} (Equation 3.25). A natural question arises: *When the optimal policy is not reached, does the η_{sub} still yield insights about the exploratory process of RL?* In Table 3.4, we compare ESL when the optimal policy was reached (η), versus when it was not (η_{sub}).

We observe that η_{sub} values are always greater than η values. However, they both yield the same efficiency ranking (e.g. PSRL, UCRL2, SAC and DQN). This indicates that η_{sub} reliably predicts results provided by η for relative comparison of algorithms.

Table 3.4: *Evaluation of algorithms in the* ***stochastic, dense-rewards setting*** *for the gridworld. When the algorithm converged at optimality, η is the Effort of Sequential Learning, and $d = \mathcal{W}_1(v_{\pi_0}, v_{\pi^*})$ is the distance between the initial and optimal policies. When the algorithm did not converge at the optimal policy but some π_N, we use η_{sub} and $c = \mathcal{W}_1(v_{\pi_0}, v_{\pi_N})$ to denote the aforementioned quantities.* 40 *training trials were used.*

Algo.	η	η_{sub}	d	c
SAC	445±246	853±127	5.63±1.23	7.26±1.45
UCRL2	198±121	510±274	5.36±0.84	4.58±1.90
PSRL	55.4±33.6	361±43.6	4.97±1.34	3.91±0.48
DQN	458±311	1971±250	4.88±1.06	6.52±0.31

3.6.4 ESL Increases with Task Difficulty

To study the effects of task difficulty on ESL, we assess five different tasks in 2D-Gridworld environments shown in Figure 3.7. The configurations of the five tasks are all deterministic with actions: {up, right, down, left}, and mostly have the start-state at the top-left and the goal-state at the bottom-right - with only one task that has the goal-state at the center. In the order of appearance: a) $[5 \times 5]$ *dense*: has size 5×5 and dense rewards; b) $[5 \times 5]$ *sparse (hard)*: has size 5×5 and sparse rewards; c) $[5 \times 5]$ *sparse (easy)*: has size 5×5, sparse rewards and goal-state at the center; d) $[15 \times 15]$ *dense*: has size 15×15 and dense rewards; and e) $[15 \times 15]$ *sparse*: has size 15×15 and sparse rewards. The reward functions for both dense and sparse rewards are as described in the previous sections.

Figure 3.8 depicts the ESL values for Q-learning with ϵ-decay strategy (for $\epsilon = 0.9$) across the five gridworld tasks with varying hardness. We chose to assess the ϵ-decay Q-learning algorithm because it is simple and yet completes all these tasks successfully. Using heuristics such as (a) the size of the state space, (b) reward formulation (i.e. dense vs sparse) and (c) proximity of the start and goal states, we rank the tasks according to their hardness as follows (easiest to hardest): 1. $[5 \times 5]$ *dense*, 2. $[5 \times 5]$ *sparse (easy)*, 3. $[5 \times 5]$ *sparse (hard)*, 4. $[15 \times 15]$ *dense*, and 5. $[15 \times 15]$ *sparse*. This ranking assumes that the task hardness of the environments scales with the size of the state space (Conserva & Rauber, 2022) and the distance between the start and goal states. Furthermore, tasks with sparse rewards are taken to be harder than those with dense rewards (Pathak et al., 2017).

We observe that the ESL is lowest for $[5 \times 5]$ *dense* (easiest task) and highest for $[15 \times 15]$ *sparse* (hardest task). The results demonstrate that ESL scales proportionally with task difficulty, matching expectations that more difficult tasks demand greater effort of the exploratory process.

Remark 4. *When the optimal policy is reached, we can use the visualization of policy trajectories, as well as ESL (η) and OMR to study the exploratory processes of an RL algorithm. When the optimal policy is not reached, we can still use the visualization*

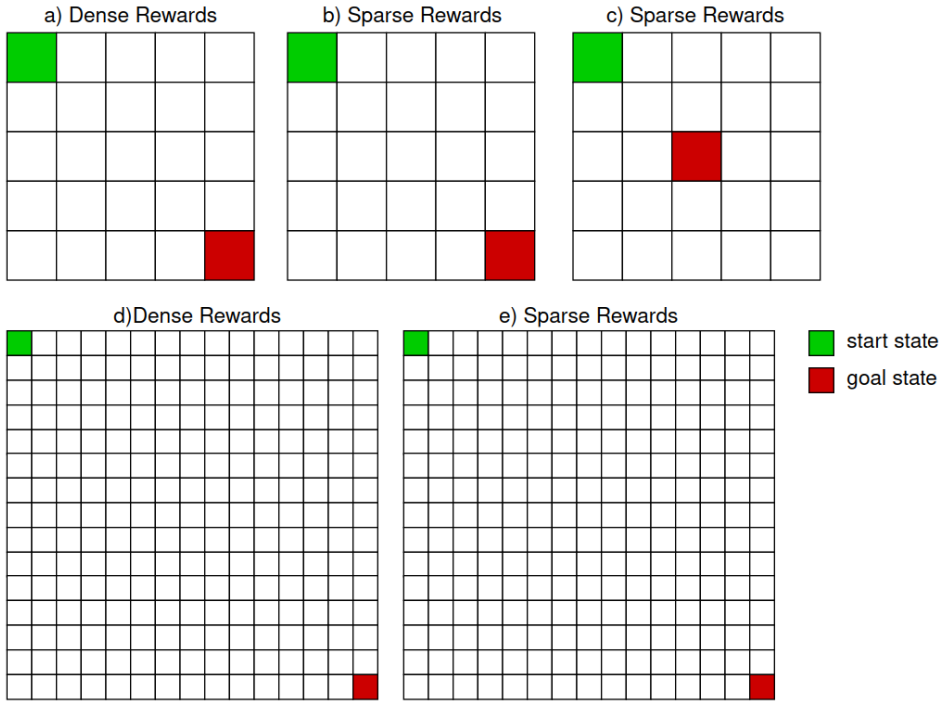


Figure 3.7: *Five gridworld tasks with the same action space, but different rewards, state space and location of the goal state.*

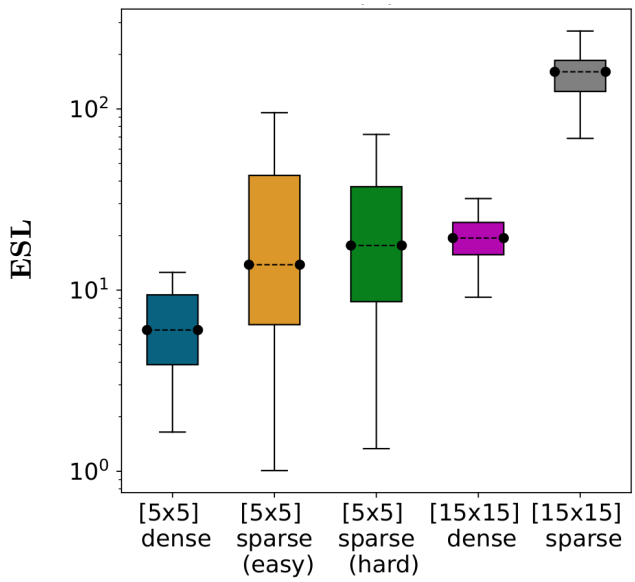


Figure 3.8: *Q-learning with ϵ-greedy ($\epsilon = 0.9$ decaying, averaged over 40 runs) across deterministic 2D-Gridworld (5×5 and 15×15) tasks. The 1st and 4th (from left to right) have dense rewards, while the rest have sparse rewards.*

of policy trajectories and ESL-sub (η_{sub}) to study the exploratory processes because they still capture characteristics of exploratory processes (e.g. high coverage, smooth exploration). Additionally, ESL (η or η_{sub}) captures the hardness of the task that we are solving. Thus, studying the occupancy measure trajectories and their corresponding indices can aid in making a knowledgeable choice of an RL algorithm that exhibits desired characteristics.

3.7 Related Works

Several prior works have utilized various components leveraged in our work, namely Wasserstein distance, occupancy measures, and the trajectory of RL on a manifold, but for different purposes. Here, we summarise them and elucidate the connections.

In supervised learning, Alvarez-Melis & Fusi (2020) proposed an optimal transport approach, namely Optimal Transport Dataset Distance (OTDD), to quantify the transferability between two supervised learning tasks by computing the similarity (i.e. distance) between the task datasets. Here, we conceptualise and define the effort of learning for RL, as a sequence of such supervised learning tasks. We observe that *the total effort of sequential learning can be computed as the sum of OTDD distances between consecutive occupancy measures*. Recently, Zhu et al. (2024) have developed generalized occupancy models by defining cumulative features that are transferable across tasks.

Optimal transport-based approaches are also explored in RL literature. These works broadly belong to two families. First line of works uses Wasserstein distance over a posterior distribution of Q-values (Metelli et al., 2019; Likmeta et al., 2023) or return distributions (Sun et al., 2022) to quantify uncertainty, and then to use this Wasserstein distance as a loss to learn better models of the posterior distribution of Q-values or return distributions, respectively. The second line of works uses Wasserstein distance between a feasible family of MDPs as an additional robustness constraint to design robust RL algorithms (Abdullah et al., 2019; Derman & Mannor, 2020; Hou et al., 2020). Here, *we introduce a concept of using Wasserstein distance between occupancy measures to understand the exploratory dynamics*. Incorporating this insight into better algorithm design would be an interesting future work. Recently, Calo et al. (2024) related Wasserstein distance between reward-labelled Markov chains to bisimulation metrics which abstract state spaces. In the same spirit, we could use reward as the cost-function in computing our nested Wasserstein distance (OTDD) to obtain a reward- or value-aware OTDD to define broader bisimulation metrics with abstract state-action spaces, instead of just state spaces.

As a parallel approach to optimal transport, the information geometries of the trajectory of an RL algorithm under different settings are studied. These approaches use mutual information as a metric instead of Wasserstein distance. Basu et al. (2020) studied the information geometry of Bayesian multi-armed bandit algorithms. They considered a bandit algorithm as a trajectory on a belief-reward manifold, and proposed

a geometric approach to design a near-optimal Bayesian bandit algorithm. Eysenbach et al. (2021) and Laskin et al. (2022) studies information geometry of unsupervised RL and proposed mutual information maximization schemes over a set of tasks and their marginal state distributions. Yang et al. (2024) extended this approach with Wasserstein distance and demonstrated benefits of using Wasserstein distance than mutual information. *We use Wasserstein distance as a natural metric in occupancy manifold which aligns with the hardness of different tasks.* It would be interesting to extend our framework to understand the dynamics of unsupervised RL algorithms.

3.8 Discussion

Our work introduces methods to theoretically and quantitatively understand and compare the exploratory strategies of different RL algorithms. Since learning in a typical RL algorithm happens through a sequence of policy updates, we propose to understand the exploratory process by visualizing and analyzing the path traversed by an RL algorithm in the space of occupancy measures. We show the usefulness of this approach by conducting experiments on various environments and RL algorithms.

Our results show that ESL and OMR provide insight into the evolution of the agent's policy, revealing whether it is approaching the optimal policy in a steady or meandering manner. Additionally, they allow us to understand how the learning process of the same algorithm changes with different rewards and transitions structures, and task hardness. We emphasize that ESL and OMR complement the number of updates to converge and regret rather than replace them as follows:

1. Complementarity of ESL and OMR with respect to number of updates:

(a) **Case 1.** Let us consider two RL algorithms that reach optimality with the same number of updates, i.e. they have the same UC. *How would one be able to distinguish the exploratory processes of these algorithms?* ESL and OMR are the summary metrics of the policy trajectory during learning. These can reveal which algorithm's exploratory process is more direct versus meandering, smooth versus noisy, or has large versus small coverage area in the policy space (Figures 3.5 and 3.6, top rows). Therefore, ESL and OMR quantify with granularity the characteristics of the exploratory process of an RL algorithm for any given environment.

(b) **Case 2.** Let us consider the case when optimality is not reached but the maximum number of updates is attained by two RL algorithms. *How would one be able to evaluate the exploratory processes of these algorithms and systematically uncover which exploratory process demonstrates desired characteristics?* Looking into the training trajectories of RL algorithms in an environment and corresponding higher/lower ESLs (η_{sub}, Section 3.5.2), we can make a knowledgeable choice of an RL algorithm exhibiting desired characteristics (e.g. high coverage, smooth exploration). We have shown in Section 3.5.2 and results in Section 3.6.3 (also Ap-

pendix B.1) that ranking based on suboptimal ESL is aligned with true ESL, and additionally, the visualization of the training trajectories (Figures 3.5 and 3.6) can indicate the characteristics of corresponding RL algorithms even when optimal policy is not reached.

(c) **Experimental Evidence.** UCRL2 is known to be provably regret-optimal and is designed to continuously explore. SAC does not have such rigorous theoretical guarantees but is known to be practically efficient. In Table 3.1, by UC, we observe that SAC is significantly suboptimal than UCRL2. But SAC has lower ESL than UCRL2 as its exploration is smoother. Additionally, OMR for SAC is higher than that of UCRL2. They together indicate that SAC takes smoother but larger number of policy transitions aligned to optimal direction for exploration, while UCRL2 exhibits bigger policy changes and in diverse manner trying to cover the environment faster.

2. Complementarity of ESL and OMR with respect to Regret: UCRL2 and PSRL have the same order of regret bounds (Osband et al., 2013). But PSRL leads to smoother policy transitions that are much more orientated towards optimality (as shown in Figure 3.5), while UCRL2 leads to less smooth policy transitions that do not taper as it approaches optimality. This information is not evident from regret but from corresponding ESL and OMR values in Table 3.1.

3.8.1 Practical Aspects

This section discusses practical features of using this framework.

Computational complexity. While efficiency is an important aspect, the primary focus of this work is to introduce a framework for analysing exploration in RL using occupancy measures. It is of interest to utilise our approach to benchmark and compare the learning dynamics of various RL algorithms in more environments, especially large-scale or high-dimensional ones. Computing Wasserstein distances in such environments would incur high computational costs, however in recent years several methods such as greedy computation (Carlier et al., 2010), hierarchical methods (Lee et al., 2019), and inexact proximal point methods (Xie et al., 2020) have been introduced to handle large-scale OT problems. For example, Gao & Chaudhari (2021) leverages a block-diagonal approximation method to deal with high-dimensional probability distributions similar to ours, while anchor space OT (Huang et al., 2024) specifically addresses multiple OT problems with multiple distributions.

Sensitivity of ESL and OMR to downsampling resolution. ESL and OMR are dependent on computing the curve length traversed by the algorithm in the occupancy measure space during training. It is crucial to utilise all policies encountered by the algorithm when determining the curve length (including ESL and OMR). Using fewer policies, i.e. through downsampling, to estimate the curve length may improve computational efficiency but compromise reliability. The reason is that downsampling introduces a systematic underestimation bias in the curve length estimation. For ex-

ample, suppose *algorithm A* has a very erratic and meandering exploration path while *algorithm B* has a smoother and more direct path. With downsampling, ignoring oscillations in between the sampled points (policies) may lead to the underestimation of the distance travelled by *algorithm A*. However, since the path for *algorithm B* is nearly straight, fewer points will still produce high accuracy approximation. Therefore to avoid misleading results, all policies encountered by the algorithm during training must be used to compute ESL and OMR.

Choice of distance metric $d_{\mathcal{X}}$. The choice of the distance metric impacts the geometry of the support space (Lee, 2009), consequently the Wasserstein distances, and thus ESL and OMR. In our case, the support space (i.e. state-action space) is reduced to the state space, as the action space maps back to the state space via OTDD. Thus, the choice of distance metric should reflect the effort of moving in the state space. For example, in the Gridworld, we used $L1$ distance because only vertical and horizontal displacements are allowed, and $L2$ distance in the Mountain Car, as applicable to real-world spaces.

Algorithm Selection and Design. Depending on the environment, we can select an algorithm with promising characteristics and spend more time optimising it to improve performance. For example, rather than tuning the hyperparameters of multiple competing algorithms to find the best one, it may be more effective to first identify an algorithm best suited to an environment based on ESL and OMR, and then fine-tune it. The chosen algorithm might remain suitable across similar environments as well.

Furthermore, we could incorporate an online adaptation of the exploratory process of the RL algorithm itself, based on recent estimates of the ESL and OMR. For example, if the current policy gives a better return than the initial policy, then we could adapt the exploratory parameters (at a slow rate) to optimise a running estimate of η_{sub} and a suitable approximation of OMR, thus enabling better exploration. However, the feasibility and convergence of such a scheme remains open.

3.8.2 Concluding Remarks

This chapter introduced an evaluation framework of RL algorithms with respect to exploration. With this established, we can proceed to study evaluation frameworks for task complexity and robustness. Recall that task complexity and robustness can provide basis for understanding generalisability as they significantly impact it (Xu & Mannor, 2012; Justesen et al., 2018). Moreover, since generalisation can be challenging in RL to assess directly (Kirk et al., 2023), we indirectly study it via studying robustness and task complexity — in a similar approach followed by Kuzborskij & Lampert (2018). We begin with task complexity in the next chapter, followed by robustness in subsequent chapters.

Chapter 4

Task Complexity

This chapter is taken from the following article (also referred to as (Nkhumise et al., 2026)):

Reabetswe M. Nkhumise, Mohamed S. Talamali, Aditya Gilra. "Issues with measuring task complexity via random policies in robotic tasks." Proceedings of the 25th International Conference on Autonomous Agents and Multi-Agent, May 2026.

4.1 abstract

Reinforcement learning (RL) has enabled major advances in fields such as robotics and natural language processing. A key challenge in RL is measuring task complexity, which is essential for creating meaningful benchmarks and designing effective curricula. While there are numerous well-established metrics for assessing task complexity in tabular settings, relatively few exist in non-tabular domains. These include (i) Statistical analysis of the performance of random policies via random weight guessing (RWG), and (ii) information-theoretic metrics Policy Information Capacity (PIC) and Policy-Optimal Information Capacity (POIC) which are reliant on RWG. In this chapter, we evaluate these methods using progressively difficult robotic manipulation setups, with known relative complexity, with both dense and sparse reward formulations.

Our empirical results reveal that measuring complexity is still nuanced. Specifically, under the same reward formulation, PIC suggests that a two-link robotic arm setup is easier than a single-link setup — which contradicts the robotic control and empirical RL perspective whereby the two-link setup is inherently more complex. Likewise, for the same setup, POIC estimates that tasks with sparse rewards are easier than those with dense rewards. Thus, we show that both PIC and POIC contradict typical understanding and empirical results from RL. These findings highlight the need to move beyond RWG-based metrics towards better metrics that can more reliably capture task complexity in non-tabular RL with our task framework as a starting point.

4.2 Introduction

Novel RL algorithms are continually being developed to tackle increasingly complex real-world problems. To evaluate progress, researchers employ a variety of RL *benchmarks* (Bellemare et al., 2013; Tassa et al., 2018; Osband et al., 2019) — collections of tasks designed with varying levels of difficulty — to assess and compare algorithms (Oller et al., 2020). When proposing a new method, it is standard practice to use such benchmarks to characterise its capabilities relative to existing approaches by measuring performance across tasks and examining how results scale with increasing difficulty. In addition, researchers also often leverage *curriculum learning*, where agents are trained on a sequence of progressively more difficult tasks to enable intermediate learning and gradual skills acquisition (Narvekar, 2021). As highlighted in Section 1.3, this approach has been shown to improve both learning and generalisability (Justesen et al., 2018; Cobbe et al., 2019; Rajan et al., 2023), with agents trained using curricula typically outperforming those trained directly on the most difficult tasks (Bengio et al., 2009; Justesen et al., 2018; Narvekar, 2021).

Measuring task difficulty is essential in both benchmarks and curriculum learning. In benchmarks, it ensures that tasks span a broad spectrum of challenges, providing comprehensive coverage that avoids sets that are uniformly trivial or overly difficult. This enables a more rigorous evaluation of algorithms' capabilities (Duan et al., 2016; Rajan et al., 2023). In curriculum learning, by contrast, task difficulty supports the structured ranking of tasks, allowing agents to encounter them in a progression that reflects their true hardness (Narvekar et al., 2020).

RL tasks are commonly divided into tabular and non-tabular settings, as mentioned in Section 2.1. Tabular RL assumes small, finite state-action spaces that can be explicitly enumerated, as in the grid-world or bandit problems. Non-tabular RL, by contrast, involves large or continuous state-action spaces, typical of robotics control, autonomous driving, or energy management (Kormushev et al., 2013; Dulac-Arnold et al., 2020; Rocchetta et al., 2019; Furuta et al., 2021). While there are well-established metrics of task difficulty in tabular settings (Abel et al., 2021; Conserva & Rauber, 2022), there still lacks a unified task complexity framework in non-tabular tasks (Conserva & Rauber, 2022); as most existing approaches rely on heuristics, impose restrictive assumptions or are computationally intractable (Conserva & Rauber, 2022; Conserva et al., 2025).

Nevertheless, two notable approaches have been proposed to broadly analyse task complexity in non-tabular domains. Both are based on the Random Weight Guessing (RWG) (Schmidhuber et al., 1999) process. In this process, untrained policies (initialised with random weights) are executed within tasks and their cumulative rewards (returns) are measured (Oller et al., 2020). The first approach employs *statistical analysis* of the resulting return distributions to assess task difficulty (Oller et al., 2020). The second approach adopts an *information-theoretic* perspective (Murphy, 2012), introducing two metrics: *Policy Information Capacity* (PIC) and *Policy-Optimal Infor-*

mation Capacity (POIC) (Furuta et al., 2021).

PIC quantifies the mutual information between the policy weights (parameters) and the returns. In contrast, POIC measures the mutual information between the policy parameters and an optimality variable, which indicates whether the agent behaves optimally throughout the episode (Furuta et al., 2021). A higher PIC value reflects a stronger dependence of returns on the random policy parameters, suggesting that the policy exerts a greater influence on performance and that the task is therefore easier. Similarly, high POIC values indicate that finding an optimal policy is relatively straightforward. Conversely, lower PIC and POIC values are associated with more difficult tasks.

While the statistical approach provides a *relative* measure of task complexity — indicating, for instance, that one task is more difficult than another based on the return statistics of randomly sampled policies — it does not quantify *how harder* one task is compared to another. This limitation is addressed by the information-theoretic approach through the PIC and POIC metrics, which offer quantitative measures of relative task complexity.

In this chapter, we demonstrate that, despite the information-theoretic approach providing a more quantitative characterisation of relative task complexity, the resulting measures can be misleading in specific cases. Using tasks with *known relative complexity relationships*, we show that PIC and POIC can incorrectly capture task hardness. These tasks consist of simple robotic manipulation environments — *1-link* and *2-link* manipulators with one and two degrees of freedom (DoF), respectively — where the objective is to control a robotic arm to reach specified target positions. Two reward formulations are considered: a *dense* formulation, which provides incremental rewards as the arm approaches the target, and a *sparse* formulation, which provides a non-negative reward only upon reaching the target.

In our experiments, both PIC and POIC produced results that contradict expectations. For instance, the metrics suggested that the *2-link* manipulator task is easier than the *1-link* manipulator task under the same reward formulation, or that it is easier to find optimal policies in a sparse-reward setting than in a dense-reward one. These outcomes show that the PIC and POIC metrics may not be reliable indicators of task complexity, and that the question of reliable metrics remains open. We speculate that the inconsistencies of these metrics could be attributed to their reliance on the RWG process, which is known to be ineffective for tasks with sparse solution regions in the weight (parameter) space (Schmidhuber et al., 1999).

Our contributions are threefold:

- We propose a framework for assessing task complexity metrics, achieved by using environments and reward formulations of known relative complexity. Specifically, we employ structurally comparable robotic manipulation environments evaluated under different reward formulations.

- Using this framework, we show that PIC and POIC yield results that contradict these known complexity relationships, suggesting that these metrics do not remain valid in certain task settings.
- We highlight the need for continued research into developing more reliable and interpretable measures of task complexity.

The remainder of this chapter is organised as follows: we briefly outline the methods used to assess task complexity — i.e. statistical analysis of return distributions of random policies, and PIC and POIC — in Section 4.3. Following this, we explicate the complexity of manipulation tasks in Section 4.4. Then, we present the results of Task Complexity Analysis using the aforementioned methods on manipulation tasks in Section 4.5. Finally, we discuss the limitations of these task complexity metrics in Section 4.6.

4.3 Task Complexity Quantification Frameworks

In this section, we review the process used to determine the return statistics of random policies obtained through RWG, and the PIC and POIC task complexity metrics. Henceforth, we use the term *return* interchangeably with *performance*.

4.3.1 RWG and Statistical analysis

The use of RWG in RL, along with statistical analysis of performance, for analysing RL task complexity was introduced in Oller et al. (2020). In this method, a policy model is represented by a neural network architecture. The model's parameters are randomly sampled at the beginning of each run and remain fixed thereafter. The untrained policy is then executed within the environment, and the resulting episodic rewards are recorded — as outlined in Algorithm 2.

Algorithm 2: Task Evaluation with RWG

Input: Prior distribution of parameters $p(\theta) = \mathcal{N}(0, I)$, Number of samples N, Number of episodes M.

Output: episodic cumulative reward $S_{n,e}$

```
Initialize environment;
Create array S_{n,e} of size N × M;
for n = 1, 2, ..., N do
    Sample weights θ_n ~ p(θ);
    for e = 1, 2, ..., M do
        Reset the environment;
        Run episode with θ_n;
        Store cumulative episode reward in S_{n,e};
```

The prior distribution of parameters $p(\theta)$ is a multivariate normal distribution $\mathcal{N}(0, I)$, where $I \in \mathbb{R}^{d \times d}$ is an identity matrix, over weight vectors $\theta_n \in \mathbb{R}^d$. N is the number of parameter sets of the policy model, i.e. number of (random) policies. M is the number of episodes per run. Performance of each policy indexed by n is aggregated by computing the mean M_n and variance V_n of the cumulative rewards over its trial set of episodes, using:

$$M_n = \frac{1}{M} \sum_{e=1}^{M} S_{n,e} \tag{4.1}$$

$$V_n = \frac{1}{M-1} \sum_{e=1}^{M} (S_{n,e} - M_n)^2 \tag{4.2}$$

where $S_{n,e}$ is the episodic cumulative reward (i.e. performance sample) for e^{th} episode of n^{th} policy. For a given task environment, the aggregate performance of the policies are showcased in three plots:

1. Log-scale histogram of M_n
2. Mean performance M_n vs rank R_n
3. Variance performance V_n vs mean performance M_n

where rank R_n sorts the policies according to performance, with 1 denoting the policy with the lowest mean performance and larger values representing policies with higher mean performances. If two policies rank the same, then the tie is broken by ranking them in the order which their weights were sampled.

4.3.2 PIC and POIC

Mutual information is a quantity that measures the dependency between two random variables (Cover & Thomas, 2006; Murphy, 2012). Unlike the correlation coefficient, it is not limited to only linear relationships but can also describe nonlinear ones (Murphy, 2012). PIC is the mutual information between the policy model parameters and the corresponding episodic cumulative rewards (i.e. *return* samples). It is given by,

$$\mathbb{I}(R; \Theta) = \mathbb{H}(R) - \mathbb{E}_{p(\theta)}[\mathbb{H}(R|\Theta = \theta)] \tag{4.3}$$

where $\mathbb{H}(\cdot)$ is Shannon entropy, R is the episodic cumulative reward random variable, and Θ is the random variable of policy model parameters. Intuitively, if the parameters Θ do not tightly determine R (i.e. have little effect on the reward signal), then the first term and second term in Equation 4.3 will be approximately equal. That is, $\mathbb{H}(R) \approx \mathbb{E}_{p(\theta)}[\mathbb{H}(R|\Theta = \theta)]$ and hence PIC ≈ 0. In that case, the task is relatively hard and therefore, PIC $\to 0$ as tasks become harder.

POIC is the mutual information between the policy model parameters and the *optimality variable*. An optimality variable represents whether the agent behaves optimally during the entire episode (Furuta et al., 2021). For instance, when POIC is expressed as,

$$\mathbb{I}(\mathbb{O};\Theta) = \mathbb{H}(\mathbb{O}) - \mathbb{E}_{p(\theta)}[\mathbb{H}(\mathbb{O}|\Theta=\theta)] \tag{4.4}$$

$\mathbb{O}$ is the optimality variable which $\mathbb{O}=1$ when the agent behaves optimally during the episode, and $\mathbb{O}=0$ otherwise. If parameters Θ have significant effect on the agent's optimal performance, then the difference between the first and second terms in Equation 4.4 will be large. This would highlight the ease of acting optimally in the task.

Note that PIC and POIC are aligned. They respectively represent the influence of the policy model parameters Θ on rewards and optimal behaviour. Furthermore, they can be viewed as the remaining randomness in rewards or optimality after accounting for the randomness caused by the parameters Θ. The residual randomness reflects variability inherent to the environment. Both PIC and POIC make use of performance samples generated via Algorithm 2. In practice, PIC and POIC are empirically estimated by discretizing the return distribution $p(R)$ and each conditional distribution $p(R|\theta_i)$ into B identical bins, as follows — starting with PIC:

$$\begin{aligned}\hat{I}(R;\theta) = &-\sum_{b=1}^{B}\hat{p}(R_b)\log(\hat{p}(R_b)) \\ &+\frac{1}{N}\sum_{n=1}^{N}\sum_{b=1}^{B}\hat{p}(R_b|\theta_n)\log(\hat{p}(R_b|\theta_n))\end{aligned} \tag{4.5}$$

where N is number of random policies and $\hat{p}(R_b)$ estimates a portion of return samples in bin b with respect to the total number of return samples. For a given θ_n, the fraction of return samples in bin b relative to all return samples is $\hat{p}(R_b|\theta_n)$. POIC is estimated using:

$$\begin{aligned}\hat{I}(O;\theta) = &-\hat{p}_1\log(\hat{p}_1) - (1-\hat{p}_1)\log(1-\hat{p}_1) \\ &+\frac{1}{N}\left(\sum_{n=1}^{N}\hat{p}_{1n}\log(\hat{p}_{1n}) + (1-\hat{p}_{1n})\log(1-\hat{p}_{1n})\right)\end{aligned} \tag{4.6}$$

where $\hat{p}_1 \doteq p(\mathbb{O}=1) \approx \frac{1}{N}\sum_{n=1}^{N}\hat{p}_{1n}$ and $\hat{p}_{1n} \doteq p(\mathbb{O}=1|\theta_n) \approx \frac{1}{M}\sum_{e=1}^{M}\exp\left(\frac{S_{n,e}-S_{max}}{\eta}\right)$. Note that η is a tuning temperature and $S_{max} = \max[S_{n,e}, S^*]$, where S^* is the episodic cumulative reward of an optimal policy π^*. Minimum and maximum values in the

return samples are set as limits and divided into B equal parts for the calculations. In the next section, we consider the complexity of robotic reaching tasks that we will use to evaluate the task complexity metrics presented in this section.

4.4 Complexity of robotic reaching tasks

Robot manipulation is a classic problem that has been studied rigorously in control theory (Siciliano et al., 2008; Mason, 2018). Within manipulation, reaching tasks are defined by the objective of moving the end-effector to desired positions. We consider such tasks along with fully actuated serial manipulators shown in Figure 4.1, where the motion of each joint is directly controllable. A manipulator with n number of links or joints, often referred to as an *n-link* arm, has the dynamic model (Corke, 2011):

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + F(\dot{q}) + J(q)^T\eta = \tau \tag{4.7}$$

where $q, \dot{q}, \ddot{q} \in \mathbb{R}^n$ are joint position, velocity and acceleration vectors, respectively. $\tau \in \mathbb{R}^n$ is actuator joint torque vector, $M(q) \in \mathbb{R}^{n\times n}$ is inertia matrix, $C(q,\dot{q})\dot{q} \in \mathbb{R}^n$ is Coriolis and centrifugal vector, $G(q) \in \mathbb{R}^n$ is gravity vector, and $F(\dot{q}) \in \mathbb{R}^n$ is friction torque. $J(q)^T$ is the transpose of the Jacobian matrix that relates $\eta \in \mathbb{R}^6$ forces at the end-effector to joint torques.

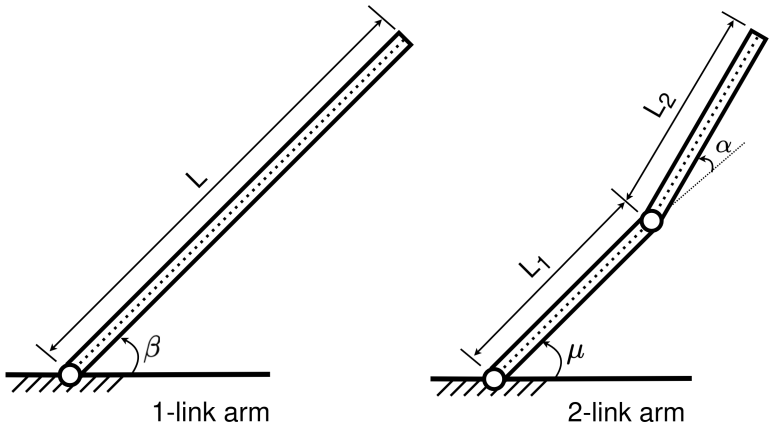


Figure 4.1: *Illustration of manipulators.*

Note that a fully actuated *n-link* arm has n degrees-of-freedom (DoF), i.e. n-DoF. Higher n-DoF enable agile, precise and energy-efficient robot motions (Murray et al., 2017), but consist of larger state-action space dimensionality. This leads to more complex dynamics (Copot et al., 2018) that contribute to the task complexity (Hentout et al., 2023). For instance, as n increases the system characteristics are impacted as follows:

1. The number of coupled terms in $M(q)$, $C(q,\dot{q})$, $F(\dot{q})$ and $G(q)$ increases (Murray et al., 2017). This means the robot dynamics become highly nonlinear and more

complex, resulting in unpredictable behaviour under perturbations (Spong et al., 2006; Sciavicco & Siciliano, 2012).

2. The coordination of multiple joints becomes more intricate and necessary to avoid factors such as link collisions, joint limits and singularity (Chiaverini, 2002; Nakanishi et al., 2008; Siciliano et al., 2009). This means controlling the system becomes more difficult.

3. The ability of the robot to move in arbitrary directions, called manipulability, increases (Yoshikawa, 1985; Vahrenkamp et al., 2012; Khadem et al., 2018). This means the set of possible target positions for the end-effector grows. This comes with high computational control effort (Nakanishi et al., 2008), since the algorithms are burdened with learning more target positions.

Solving Equation 4.7 to compute joint motion requires algorithm complexity of $\mathcal{O}(n)$ (Featherstone, 2008). This illustrates how the computational costs of the dynamics scale with n-DoF. In general, it be can declared that controlling an *n+1-link* manipulator is inherently more difficult than controlling an *n-link* manipulator. In summary,

$$\mathcal{C}(\textit{n-link manipulator}) < \mathcal{C}(\textit{n+1-link manipulator}) \tag{4.8}$$

where $\mathcal{C}(\cdot)$ denotes the hardness of controlling the system. Although Equation 4.8 is not quantitative, it provides a consistency condition against which later quantitative estimates can be assessed. In the next section, ranking of tasks via Equation 4.8 based on complexity of manipulation, will be compared with those provided by the task complexity metrics outlined in Section 4.3.

4.5 Experimental evaluation

In this section, the methods described in Section 4.3 are evaluated in a class of reaching tasks. The purpose of the experiments is to answer the following questions:

1. *Can the statistical analysis of the performance of random policies and PIC/POIC effectively capture task complexity of reaching tasks?*

2. *Do task difficulty levels ranked by PIC/POIC align with the ranking suggested by Equation 4.8?*

Section 4.5.1 describes the experimental setup, while Section 4.5.2 introduces the task framework used for assessing the accuracy of the task complexity metrics presented in Section 4.3. In Section 4.5.3, we train RL agents on the tasks defined within this framework and use their performance to verify the complexity of the tasks. The resulting measures are then compared with the known complexity rankings from robotics theory. Finally, Section 4.5.4 examines the limitations of the task complexity metrics, showing that PIC and POIC are inaccurate measures of task complexity.

4.5.1 Experimental Setup

We employ manipulators shown in Figure 4.1 in six task settings (discussed in Section 4.5.2). For each task, both the end-effector and its target position are randomly initialised at the start of each episode. This is a good training practice that prevents environment overfitting when training RL agents (Whiteson et al., 2011). We ensure reachability of target positions by sampling them exclusively from each manipulator's workspace. Friction is ignored, states are assumed to be fully observable, and the arms are operated on a horizontal plane; hence, the effects of gravity are not considered.

The arm configurations are evaluated on dense- and sparse-rewards. In dense-reward settings, the reward function is

$$r = -\omega_1 \| P_{ee} - P_g \|_2^2 - \omega_2 \| action \|_2^2 \tag{4.9}$$

where $[\omega_1, \omega_2] = [1, 1]$ are distance and control weights. P_{ee} and P_g are respectively end-effector and target/goal positions. In sparse-reward settings, the reward function is

$$r = \begin{cases} 0, & \text{if } \| P_{ee} - P_g \|_2 < 0.05 \text{ meters} \\ -1, & \text{otherwise} \end{cases} \tag{4.10}$$

where 0.05 meters is the threshold distance between the end-effector and target position. All the tasks have a maximum of 50 steps per episode, 500 training episodes and $N = 10^4$ samples (i.e. random policies). To match the experimental setup in (Furuta et al., 2021), the policy network consists of 2 hidden layers with 32 neurons each and the number of discretisation bins $B = 10^5$. During experimentation, the robotic manipulators were simulated using the *Pybullet* (Coumans & Bai, 2016) physics simulator. Note that the codebase for our work in this chapter is available at: `https://github.com/nkhumise-rea/task_complexity.git`.

4.5.2 Tasks framework

Our framework consists of six tasks that include three arm setups, each with dense- and sparse-rewards. The arm setups include: (1) *1-link* arm with link length $L = 1.00$ meter, (2) *1-link* arm with link length $L = 1.65$ meters, and (3) *2-link* arm with link lengths $L_1 = 0.95$ and $L_2 = 0.70$ meters. We ensured that the link lengths of *2-link* arm sum to 1.65 m, to have the same total length as the *1-link* arm in (2) above. This makes these two arms (2) and (3) have equivalent magnitude of error at the end-effector, leading to rewards that can be directly comparable, while varying in complexity only due to the number of links/joints.

The error in the end-effector arises from the errors in the arm joint angles being amplified by the link lengths, which results in higher positional errors at the end-effector

for longer link lengths, as captured by (for small errors)

$$\|\delta x\|_2 = \left\| \sum_{k=1}^{n} J_k(\theta)\delta\theta_k \right\|_2 \leq \epsilon \sum_{i=1}^{n} i l_i \tag{4.11}$$

where δx is error at the end-effector, $\delta\theta_k$ is angle error of the $k-$th joint, and $J_k(\theta)$ is the $k-$th column of the Jacobian matrix. n is the number of DoF, while l_i is the $i-$th link on the arm. ϵ is the worst-case joint error, i.e. $|\delta\theta_k| \leq \epsilon$ for $k = 1, \cdots, n$ (see Appendix C for details on Equation 4.11). Generally, tasks with higher error rates present greater learning challenges for RL algorithms, thus resulting in reduced rewards or longer learning time (Wang et al., 2020; Edmondson & Petrick, 2025).

We selected the three arm configurations to study independently the effects of altering link lengths and number of joints. This simplifies task comparison and ensures task structural homogeneity. Following Equation 4.8, we expect *1-link* arm tasks to be easier than the *2-link* arm task — under the same reward formulation. From Equation 4.11, we expect the *1-link* arm task with a shorter link length to be easier than the *1-link* arm task with a longer link length. Additionally, we expect consistency with RL literature (Andrychowicz et al., 2017; Pathak et al., 2017; Sutton & Barto, 2018), where tasks in dense-reward settings are easier than those in sparse-reward settings.

Importance of structurally similar tasks. Compared to most RL benchmarks, often the tasks are varied (as they should) but not structurally related (Duan et al., 2016; Bellemare et al., 2013), e.g. Cartpole and MountainCar in OpenAI Gym (Brockman et al., 2016). As such, most benchmarks can be unsuitable for reliably assessing new task complexity methods. To ensure meaningful validation, methods should first be evaluated on families of closely related tasks with known relative complexity (such as our setup). This enables consistency verification, clearer interpretation of results, and easier characterisation of the proposed methods prior to applying them to large heterogeneous benchmarks. In the subsequent section, we compare RL curves across our task framework for a baseline algorithm, to confirm if these match our expectations.

4.5.3 Reinforcement learning of tasks

We compare learning curves of a state-of-the-art algorithm Soft Actor Critic (SAC) (Haarnoja et al., 2018) across the tasks in our task framework. Studying how the algorithm performs during training provides us with information about convergence, such as the optimal return and convergence time for the tasks. The learning curves are presented in Figure 4.2. Details about the SAC architecture and its hyperparameters are provided in Appendix C.2. Note that the same network model for SAC was employed in all the tasks, and moreover, we utilised *Tianshou* (Weng et al., 2022) RL library to facilitate implementation and training of the algorithm.

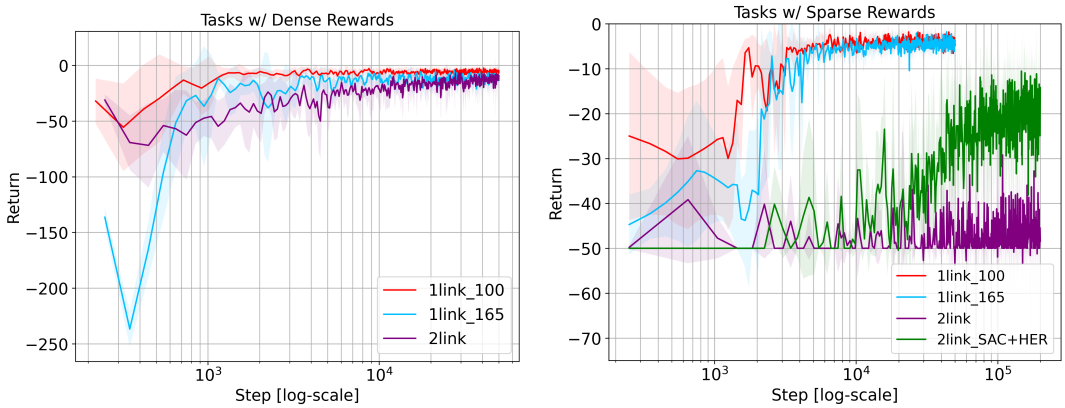


Figure 4.2: *Learning curves of SAC algorithm across the six tasks. The left panel depicts agent performance in dense-reward settings, while the right panel is in sparse-reward settings. To accommodate wide and varying ranges of steps, results are plotted on a logarithmic scale to enhance interpretability. In the 2-link arm with sparse rewards, SAC results are presented with HER (Andrychowicz et al., 2017) augmentation (SAC+HER) and without it.*

Dense-reward settings. We note in Figure 4.2 that *1-link* arm with link length $L = 1.0$ m converges faster and to a higher return than *1-link* arm with link length $L = 1.65$ m. Similarly, in the *1-link* arm with link length $L = 1.65$ m, the agent converges faster than in the *2-link* arm task. This highlights the order of tasks based on their hardness (from easiest to hardest) to be *1-link* arm ($L = 1$ m), *1-link* arm ($L = 1.65$ m) and *2-link* arm. This is consistent with our expectations as supported by Equations 4.8 and 4.11.

Sparse-reward settings. In Figure 4.2, we see that pure SAC is unable to solve the *2-link* arm task, showing that *2-link* arm task is harder than the *1-link* arm tasks, aligning with Equation 4.8. Even SAC augmented with Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) takes significantly longer to converge in the *2-link* arm task compared with *1-link* arm tasks. Further, SAC converges faster in the *1-link* arm ($L = 1$ m) task than in *1-link* arm ($L = 1.65$ m) task. This ordering of the three arms is consistent with our expectations, in this sparse-reward settings as well.

Dense-reward vs Sparse-reward settings. By comparing the dense- and sparse-reward settings, we observe that the algorithm converges quicker in dense-reward settings than in sparse-reward settings. Furthermore, SAC performance is noisier in the sparse-reward settings than its counterpart. This coincides with intuition that tasks with dense rewards are easier than with sparse rewards.

Remark 5. *In our settings, SAC(+HER) could solve the tasks (verified by demonstrations). However, in general, algorithms may fail to solve the task or perform optimally. This restricts the usage of learning curves in assessing task complexity.*

In the next section, we compare methods introduced in Section 4.3 for quantifying task

complexity. These methods are independent of specific RL algorithms.

4.5.4 Task Complexity Analysis

In this section, we categorise our results into examining how task complexity is influenced by a) link length, b) number of DoF, and c) reward formulation. We compare measures of task complexity beginning with statistical analysis of performance, and then follow up with PIC and POIC.

(I) Statistical analysis of performance. To carry out this analysis, we used Algorithm 2 to capture the cumulative rewards (returns or performance) of $N = 10^4$ randomly sampled policies via RWG (Oller et al., 2020). The performance was then aggregated into mean M_n and variance V_n using Equations 4.1 and 4.2. We visualise the aggregated performance in three plots: *mean performance histograms* (Log-scale histogram of M_n), *mean performance curve* (M_n vs R_n plot), and *variance distribution* (V_n vs M_n plot). Note that R_n is rank, from lowest to highest mean performance.

Figure 4.3 presents the performance plots, where the left, middle and right columns, respectively, depict the *mean performance histograms*, *mean performance curves*, and *variance distributions*. We normalised the mean M_n and variance V_n in each task to avoid scale-induced bias and ensure commensurability of performance (Cobbe et al., 2020; Agarwal et al., 2021). We used *min-max scaling* (Murphy, 2012),

$$x' = \frac{x - \min[x]}{\max[x] - \min[x]} \tag{4.12}$$

where x is the variable being scaled.

Overall description. We notice in Figure 4.3 that the histograms (across the tasks) have an overall shape that approximates a Gaussian distribution that does not span the entire range of mean scores. The performance curves (black curves) have smooth slopes without jumps, and the variance distributions mostly reveal that performance consistency is not uniform across the mean score range. We discuss the plots in detail below.

L = 1.0 vs L = 1.65m (*1-link*, dense-rewards). In dense-reward settings for *1-link* arms, where one has $L = 1$ m and the other has $L = 1.65$ m, we make the following observations,

Performance histograms: In both tasks, no single random policy achieved mean performance M_n near the maximum return. This exhibits that the tasks are not trivial (Oller et al., 2020). The performance distributions in both the arms are similar. This shows that the tasks are structurally equivalent. This aligns with intuition, as the link lengths impacts reward scales (see Figure 4.2), but do not alter the task structure, as seen after normalisation of rewards.

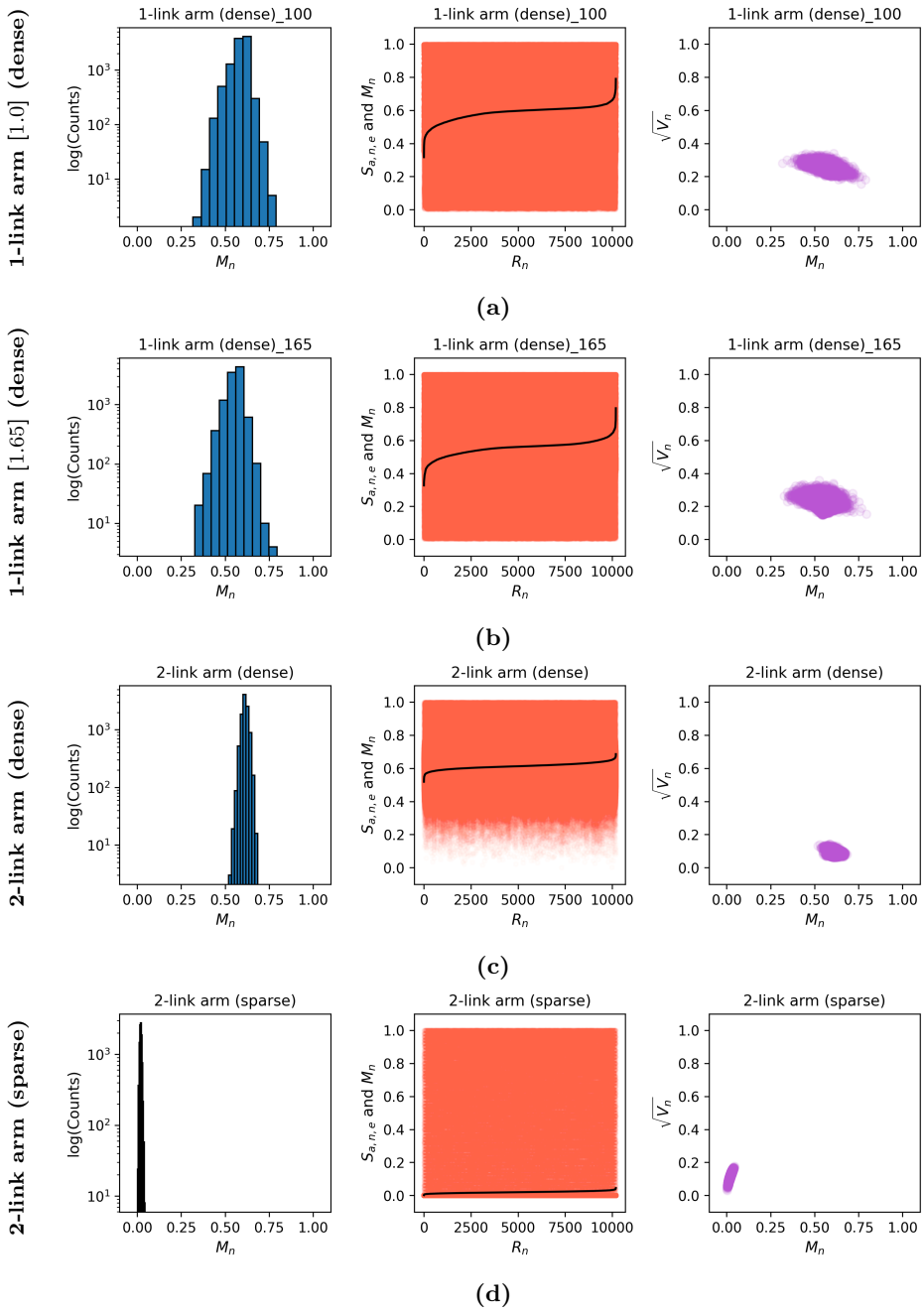


Figure 4.3: *Performance distribution plots for the tasks: (a) 1-link (L=1.0m), (b) 1-link (L=1.65m), (c) 2-link arms with dense rewards, and (d) 2-link arm with sparse rewards. The left column shows a histogram of mean performances of the random policies (Log-scale histogram of M_n). The middle column depicts mean performance curves in black, i.e. mean performance M_n vs rank R_n. Moreover, all the cumulative rewards of the policies $s_{a,n,e}$ across the trials are represented by red dots (behind the black curve). The right column displays plots of standard deviation $\sqrt{V_n}$ vs mean performance M_n (often referred to as variance distribution). The plots were made using 10^4 random policies.*

Performance curves: Interestingly in both tasks, the random policies managed to attain episodic cumulative rewards $S_{n,e}$ that are nearly the maximum return (shown by red dots at 1.0) in a few episodes. This is sensible since both the initial end-effector and target positions are random in every episode. With 500 episodes, it is likely that initial and target positions of the end-effector were in close proximity in some episodes — which simplifies the task in those episodes.

Variance distributions: Both arms have similar variance distributions, with wide spread about the middle mean score (i.e. $M_n = 0.5$) which slightly narrows towards the limits of the mean performance range — with minimal variance at the highest mean score. This indicates that majority of policies attain their mean performance by succeeding on only some episodes and failing at others, however better consistency is required to reach higher mean performance (Oller et al., 2020). Note that apart from Figure 4.3 revealing that the *1-link* arm tasks are similar, it is not clear which task is easier or harder between the two.

1-Link vs 2-Link (dense rewards). Figure 4.3 (2^{nd} and 3^{rd} rows) displays how the histogram has a narrower width for *2-link* arm (than for *1-link* arm) while the central M_n nearly remains consistent across the tasks. The performance curve in *2-link* arm has a low slope than that in the *1-link* arm task. Furthermore, the variance range of the performance in the *2-link* arm task is smaller. These denote that the *2-link* arm task is harder than the *1-link* arm task. The reason is that harder tasks often provide higher rewards only when a coherent sequence of successful actions is executed, which untrained random policies are unlikely to achieve, hence reduced variability in performance.

Dense vs Sparse rewards (2-link). Figure 4.3 (3^{rd} and 4^{th} rows) portrays a drastic drop in peak M_n and variance V_n, from dense- to sparse-reward settings. The performance curve slope further decreased (almost zero) in the sparse-reward setting compared to the dense-reward setting. This highlights a lack of diversity in the performance of the random policies. In the variance plots for the sparse-reward setting, we notice that random policies fail to succeed in the task regardless of initial conditions. We can conclude from these results that the dense-reward setting is easier than the sparse-reward setting.

Remark 6. *Although the statistical analysis and visualisation of performance provide some insights about the task characteristics and relative hardness, they fail to quantitatively measure task difficulty, i.e. the approach is qualitative. This makes it inapplicable to RL benchmarks and curriculum learning, where relative hardness amongst tasks needs to be quantified. Moreover, performance distributions that are similar across tasks can potentially make the plots less informative in comparing the tasks. For these reasons, we now examine quantitative metrics PIC and POIC.*

(II) PIC/POIC. The quantitative representations of task complexity offered by PIC and POIC are exhibited for our six tasks in Table 4.1. We checked the statistical robustness of the results in Table 4.1 by quantifying their uncertainty using bootstrapped

confidence intervals (DiCiccio & Efron, 1996). These estimate the uncertainty by repeatedly resampling the data. In our context, the data are episodic cumulative rewards of random policies constructed via RWG. We resampled the data $1,000$ times with replacement and computed the values presented in Table 4.1.

We then applied Welch's t-test (Delacre et al., 2017; West, 2021) to evaluate the statistical significance in the differences between values in Table 4.1, using the same $1,000$ resamples (see Table 4.2). Consistently in all cases, the p-values of the t-statistic of the Welch's t-test are in the orders of 10^{-5}, below a typical cut-off p-value $= 0.005$ which indicate strong statistical significance (Benjamin et al., 2018). It should be noted that the values in Table 4.1 were gathered using a policy network of 2 hidden layers, each with 32 neurons. 10^4 untrained policies were sampled from a multivariate normal prior distribution. We also confirmed that RWG sampled from different prior distributions and policy network architectures (see Appendix C.3.3) did not change our results.

Table 4.1: *PIC and POIC values with $N = 10^4$ samples (random policies). High PIC and POIC values correspond to easier tasks, while low values correspond to harder tasks.*

Rewards	Arm [dim]	PIC $(\times 10^{-3})$	POIC $(\times 10^{-3})$
Dense	1-link [1.0]	4005 ± 8.5	2.628 ± 56
	1-link [1.65]	4153 ± 8.4	4.105 ± 85
	2-link [0.95,1.7]	4200 ± 6.1	0.725 ± 11
Sparse	1-link [1.0]	85.11 ± 0.4	1.958 ± 34
	1-link [1.65]	71.21 ± 0.2	1.197 ± 31
	2-link [0.95,1.7]	45.95 ± 0.0	0.946 ± 7.9

Table 4.2: *Statistical significance between PIC and POIC values using Welch's t-test. Results are presented as: statistic (p-value). D stands for dense-rewards and S stands for sparse-rewards. Note that the negative statistic denotes the incorrect order of task hardness according to PIC or POIC.*

	1-link [1.0](D)	*1-link [1.65](D)*	*2-link [0.95,1.7](D)*	*1-link [1.0](S)*	*1-link [1.65](S)*
PIC					
1-link [1.0](D)	0.0 (1.0)	-	-	-	-
1-link [1.65](D)	-20.1 (2.3×10^{-7})	0.0 (1.0)	-	-	-
2-link [0.95,1.7](D)	-29.3 (2.9×10^{-8})	-8.6 (2.6×10^{-5})	0.0 (1.0)	-	-
1-link [1.0](S)	948 (4.2×10^{-12})	646 (2.8×10^{-11})	618 (3.4×10^{-11})	0.0 (1.0)	-
1-link [1.65](S)	955 (5.7×10^{-12})	649 (3.1×10^{-11})	621 (3.7×10^{-11})	25.4 (4.7×10^{-8})	0.0 (1.0)
2-link [0.95,1.7](S)	964 (6.9×10^{-12})	654 (3.3×10^{-11})	626 (3.9×10^{-11})	85.7 (6.5×10^{-8})	84.5 (3.3×10^{-8})
POIC					
1-link [1.0](D)	0.0 (1.0)	-	-	-	-
1-link [1.65](D)	-17.7 (1.2×10^{-7})	0.0 (1.0)	-	-	-
2-link [0.95,1.7](D)	30.8 (4.5×10^{-6})	59.7 (2.6×10^{-7})	0.0 (1.0)	-	-
1-link [1.0](S)	8.1 (4.8×10^{-5})	27.9 (3.3×10^{-9})	-24.2 (3.4×10^{-5})	0.0 (1.0)	-
1-link [1.65](S)	22.3 (4.6×10^{-6})	49.0 (6.6×10^{-8})	-21.6 (1.5×10^{-6})	14.4 (2.3×10^{-5})	0.0 (1.0)
2-link [0.95,1.7](S)	27.2 (7.5×10^{-6})	55.8 (3.4×10^{-7})	-17.9 (9.9×10^{-8})	19.9 (2.4×10^{-5})	11.3 (5.0×10^{-5})

Dense-reward settings. According to the PIC values under dense-reward settings (in Table 4.1), the *2-link* arm task is the easiest task (highest PIC) and *1-link* arm ($L = 1$ m) task is the hardest task (lowest PIC). This contradicts expectations based on Equations 4.8 and 4.11, and our empirical RL results. For instance, we showed using learning curves of trained agents that the *2-link* arm task is the hardest, while *1-link* arm ($L = 1$ m) task is the easiest. This is further corroborated by performance distributions of random policies in Figure 4.3. On the POIC side, *1-link* arm ($L = 1.65$ m) task is easier than *1-link* arm ($L = 1$ m) task. This does not align with Equation 4.11.

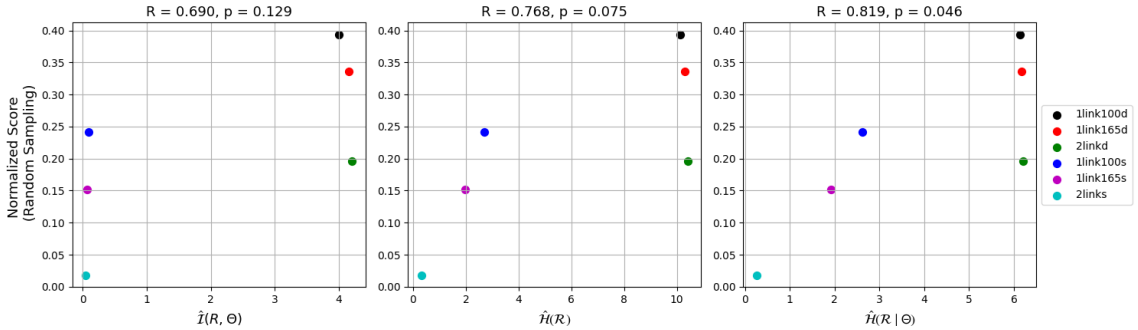
Sparse-reward settings. In these settings, both the PIC and POIC order of task difficulty across the tasks seems correct. The tasks are ordered from easiest to hardest as *1-link* arm ($L = 1$ m), *1-link* arm ($L = 1.65$ m) and *2-link* arm. This aligns with our intuition. When comparing across dense- versus sparse-reward settings, POIC values for the *2-link* arm task suggest that the dense-reward setting is harder than the sparse-reward setting. In this instance, POIC values contradict our expectations, as we showed empirically in Section 4.5.3 that dense-reward settings are easier than the sparse-reward settings. To investigate these incorrect PIC and POIC instances, we decomposed the individual entropy terms in the metrics. Figure 4.4 displays normalised scores (i.e. performance) against POIC, PIC, individual entropy terms, and the variance of cumulative rewards. The normalised scores use *min-max scaling* (Equation 4.12) over the performance samples of the random policies.

POIC related plots. The first three columns in Figure 4.4 portray POIC and entropies of the optimality variable. We observe that $\hat{\mathcal{H}}(\mathcal{O})$ and $\hat{\mathcal{H}}(\mathcal{O} \mid \Theta)$ are closely approximate, which produces POIC $\hat{\mathcal{I}}(\mathcal{O}; \Theta)$ values of small magnitude, similar to the work that introduced POIC (Furuta et al., 2021). There are multiple strong linear correlations (given by Pearson correlation coefficients) between normalised scores and the other quantities, however they are not statistically significant.

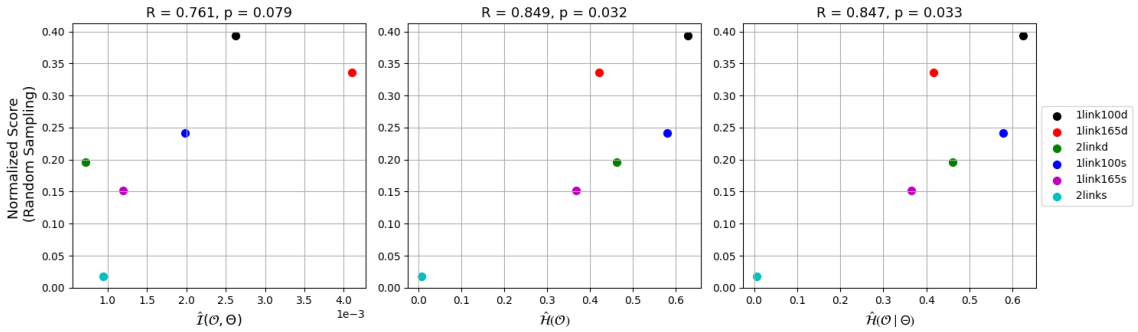
PIC related and Variance plots. In Figure 4.4 the last four columns display PIC, entropies of the cumulative reward variable, and variance of returns. We note that $\hat{\mathcal{H}}(R)$ and $\hat{\mathcal{H}}(R \mid \Theta)$ differ. This is responsible for larger magnitude values of PIC $\hat{\mathcal{I}}(R; \Theta)$. It seems dense-reward settings have more variability in returns, than sparse-reward settings. This aligns with results presented in Figure 4.3. Additionally, it seems generally that in our setup, tasks with dense-rewards enjoy higher normalised scores than sparse-rewards and this coincides with our expectations. Figure 4.4 does not provide further insights about why PIC and POIC values in Table 4.1 do not match expectations. We discuss further possibilities in the next section.

4.6 Discussion & Limitations

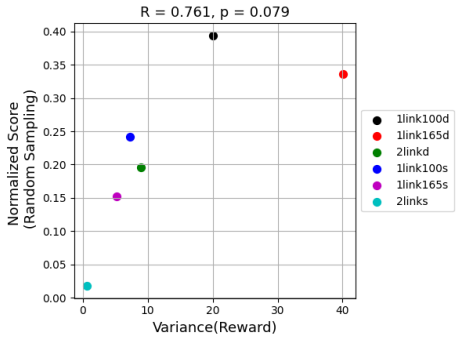
Finally, we explore potential reasons for the inconsistencies observed with RWG-based metrics PIC and POIC compared to expectations derived from robotic control and verified with empirical RL. These issues stem from (1) the dependence on randomly



(a) *PIC and related entropies*



(b) *POIC and related entropies*



(c) *Variance*

Figure 4.4: *2D-scatter plots with Normalised scores (performance) computed using min-max scaling (Equation 4.12) over the returns of untrained random policies. The Normalised scores are plotted against (a) PIC, (b) POIC, (c) variance of returns, along with entropies of optimality variable and cumulative reward (return) variable. Note that above each plot, we specified the Pearson correlation coefficient (along with the corresponding p-value) between the variables.*

generated parameters on the prior distribution $p(\theta)$, and (2) the lack of consideration for training and exploration. While statistical analysis of performance of RWG-generated policies has been consistent with intuitive expectations, it can be challenging to effectively communicate the degree of difference in task difficulty. Moreover, if tasks produce nearly similar performance distributions, then this approach might be less informative for comparative analysis.

PIC and POIC are dependent on the prior distribution of parameters $p(\theta)$, as highlighted in (Furuta et al., 2021). The prior $p(\theta)$ can be interpreted as the *effective search area* in the parameter (policy) space (Aleksandrowicz & Jaworek-Korjakowska, 2023). For problems where high-performing policies are sparsely distributed in the parameter space, the effective search area is likely to cover mainly low-performing regions. Policies sampled from these regions may yield similar actions and behaviours, ultimately exhibiting limited variability in performance (as shown in Figure 4.3). This observation reinforces the limitation of RWG originally noted by Schmidhuber et al. (1999), namely its ineffectiveness in tasks with sparse solution regions in the weight space.

Another limitation arises from the fact that RWG does not involve training. This implies that the effective search area remains static once $p(\theta)$ is selected. In contrast, training involves exploration of the policy space (Sutton & Barto, 2018; Ladosz et al., 2022; Nkhumise et al., 2025) — where the effective search area (of the learning algorithm) is dynamically moved around in the policy space. It is also important to note that neither statistical analysis of the performance of random policies, nor PIC and POIC metrics, consider the visitation complexity (Conserva & Rauber, 2022) of the tasks — which measures the difficulty in exploring the state space of the environment. This implies that the way actions influence state transitions during exploration in the learning phase is not accounted for by these task complexity methods (Furuta et al., 2021). Several methods that aim to capture exploration effort (Amin et al., 2021; Ladosz et al., 2022; Nkhumise et al., 2025) have been investigated; however, none have been applied to task complexity. This makes for an interesting future direction of work.

It is clear from our experimental results that PIC and POIC can be misleading in capturing task complexity. Inconsistencies in these metrics can be challenging to notice in most RL benchmarks, especially if they have tasks with heterogeneous structure. Our task framework offers tasks with structural homogeneity and known relative task complexity, thus enabling a more reliable assessment of these metrics.

4.6.1 Prospective Improvements & Directions

We propose the following directions for improving task complexity metrics.

Parameterised inductive bias. A key drawback of the PIC and POIC metrics is the limited effective search space sampled by RWG. We can replace the standard multilayer perceptron (MLP) policy with an architecture which introduces parameterised inductive biases relevant to the tasks. These biases are aimed at maximally covering the state-action space. By setting the parameters of these inductive biases via RWG, we can cover a wider search space. In robotics, the inductive bias can be a policy composed of dynamic movement primitives (Stulp & Schaal, 2011), skills (Dalal et al., 2021) or normalising flows (Khader et al., 2021). The drawback of this approach is that the resulting complexity measure would be dependent on the selected inductive biases.

RL Exploration. A second issue with PIC and POIC metrics is that they have a

static effective search area in the parameter space due to the lack of exploration. In devising new metrics, we want the effective search area to be dynamic by including exploration. We can perform RWG after every k updates of a RL algorithm and compute corresponding PIC/POIC values at each stage. Ultimately, use the mean PIC/POIC values across the entire learning path as the metric for task complexity. This would entail sampling weights via RWG, at every 'k' update of the policy during RL training θ_n using

$$\theta_n = \theta_k + \epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \tag{4.13}$$

where θ_k is the k^{th} policy model update. With each RWG set, we compute PIC_k, and compute the average cross the trajectory as a better measure of task complexity. The drawback of this approach is that the task complexity metric(s) would be dependent on the exploration strategy of the learning algorithm.

Known optimal policy. When the optimal policy is known, we can instead compute the distribution of distances between RWG-sampled policies and the optimal policy using optimal transport (Nkhumise et al., 2025). Task complexity could then be defined via the mean and variance of these distances, thus capturing the expected effort to move from random policies to the optimal policy. The limitation of this method is that it requires prior knowledge of the optimal policy. Note that in principle, all these three suggestions could be combined as well.

4.7 Concluding Remarks

The results presented in this chapter showcase the need for continued work in task complexity for deep RL, especially for the case of robotic tasks, where our task framework could be a starting point. Without a reliable metric for task complexity, the study of generalisation in RL (especially in curriculum learning) might be limited. This is because in curriculum learning, an effective ordering of tasks and incremental learning of the agent depends on task hardness (Narvekar et al., 2020). The agents trained with curriculum learning *perform and generalise* better those trained tabula rasa (Justesen et al., 2018; Narvekar, 2021). In our continued study of prerequisites for generalisation (viz. task complexity and robustness), we assess robustness of algorithms in the next chapter.

Chapter 5

Robustness Analysis: Hyperparameter Sensitivity and Reliability

5.1 Introduction

As covered in Section 1.3, robustness is related with the stability of the algorithm and its reliability (Paschali et al., 2018). A stable algorithm consistently performs well in the presence of slight changes in the environment (Xu & Mannor, 2012; Sun, 2015; Han et al., 2020), while a reliable algorithm has reproducible performance (Chan et al., 2020). Thus, a robust algorithm has consistent and reliable good performance across environments with close similarities (Xu & Mannor, 2012). To study robustness, we evaluate (a) the algorithm's reliability using a framework established by Chan et al. (2020), and (b) the algorithm's stability with respect to its hyperparameters. The description of algorithmic stability with respect to hyperparameters is common in literature (Henderson et al., 2018; Haarnoja et al., 2018; Wang et al., 2022; Adkins et al., 2024), hence we adopt it. We carry out hyperparameters sensitivity analysis to evaluate stability.

In summary, our analysis of robustness entails examining the algorithms' reliability (statistical behaviour) and their sensitivity to hyperparameter settings (algorithmic behaviour). The following sections provide an in-depth discussion of Reliability and Hyperparameters Sensitivity.

5.1.1 Reliability

Reliability refers to the consistency or repeatability of the algorithms' performance (Tran et al., 2022). It addresses the question of *how predictable is the algorithm's performance in the next trial?* The algorithm's reliability in RL is examined through the *dispersion* and *risk* in its performance across trials (Chan et al., 2020). Dispersion describes the

variability of performance, while risk captures how often the algorithm experiences performance drops. These measures are notably critical in robotic tasks, where high variability can inhibit accuracy and risk of failure can jeopardise repeatability (Brink et al., 2004; Lopez et al., 2019).

In robotic applications, deficiencies in accuracy and repeatability may cause production delays, safety risks, and defects (e.g. during assembly), especially in manufacturing. We thus identify robotic manipulation tasks as a compelling case study for assessing the reliability of algorithms. We particularly employ *reaching tasks*. These are a class of manipulation tasks where the objective is to move the end-effector (arm tip) to a desired position.

5.1.2 Hyperparameters Sensitivity

Hyperparameters are externally specified configuration variables that govern how an algorithm operates (Claesen & Moor, 2015; Bischl et al., 2023) — such as its learning rate, exploration-exploitation balance, neural network architecture, etc. They essentially control the algorithm's learning process (Goodfellow et al., 2016) and significantly impacts its performance (Eimer et al., 2023). RL algorithms generally consists of a set of hyperparameters (Adkins et al., 2024), which are manually tuned via rules-of-thumb or through sweeping, i.e. evaluation over a predefined set of candidate values (Claesen & Moor, 2015; Eimer et al., 2023). The tuning of hyperparameters aims to optimise algorithm performance, as different hyperparameter values tend to work best for different tasks (Bischl et al., 2023). However, hyperparameters often have an intricate interdependency that impacts the algorithm's stability (Henderson et al., 2018; Song et al., 2019). For instance, an inappropriate selection of values for a hyperparameter set can make a theoretically sound algorithm underperform, whereas proper tuning can considerably improve the performance of a suboptimal algorithm (Obando-Ceron et al., 2024).

Many RL algorithms are understood to be brittle (unstable) with respect to their hyperparameters (Haarnoja et al., 2018; Wang et al., 2022; Adkins et al., 2024). Unstable algorithms tend to be more sensitive to hyperparameter choices and demand fastidious tuning to attain good performance (Adkins et al., 2024; Patterson et al., 2024). This is a drawback, since good performance cannot be guaranteed particularly when the algorithms are employed in tasks *experiencing modest modifications* or across various tasks, *including structurally related tasks*.

For example, in robotics, it is desirable for algorithms to maintain good performance across both the training environments (e.g. simulation) and deployment environments (e.g. real-world application) (Panaganti et al., 2022; Zhang et al., 2023). In such instances, the tasks are structurally similar. Thus, algorithms requiring minimal hyperparameter tuning are an advantage since they can maintain high performance across the environments (Jordan et al., 2020; Ibarz et al., 2021). This also relates to *algorithm-level generalisation*, where good algorithm performance across tasks is achieved with

minimal hyperparameter tuning (Whiteson et al., 2011). In fact, work by Song et al. (2019) establishes that generalisation in RL is impacted by hyperparameters, notably in a non-monotonic manner.

Therefore, examining an algorithm's hyperparameter sensitivity provides the ground for understanding its stability and to some extent its generalisation. We thus perform hyperparameter sensitivity analysis of RL algorithms in reaching tasks. We adopt a framework established by Adkins et al. (2024), which defines hyperparameter sensitivity as the difference between *per-environment tuned* performance and *cross-environment tuned* performance.

5.2 Methodology

This section reviews the frameworks in RL for reliability analysis and hyperparameter sensitivity analysis, as introduced by Chan et al. (2020) and Adkins et al. (2024), respectively. Additionally, we discuss the use of prevalent sensitivity analysis methods for assessing system model behaviour in the context of hyperparameters.

5.2.1 Reliability Metrics

The performance behaviour of an RL algorithm is characterised *across runs* and *across time* (per-run) during training. While after learning, it is characterised *across rollouts*. In each dimension (i.e. across runs, time-per-run, and rollouts), dispersion and risk are employed to capture the reliability of performance. Dispersion describes the extent to which performance is spread out (around the mean or median). Risk can specify (I) the amount of performance drop (across time) and its probability of occurrence, or (II) the worst performance (across runs) and its probability of occurrence. *Inter-quartile range* (IQR) and *Conditional Value at Risk* (CVaR) are used to measure dispersion and risk, respectively.

IQR and CVaR. To understand IQR and CVaR, consider a random variable $X \sim \mathcal{N}(\mu, \sigma^2)$ whose distribution is depicted in Figure 5.1. For $\alpha \in (0, 1)$, a quantile q_α is a value such that the probability of X being at most q_α is at least α (Schervish et al., 2012), i.e.

$$q_\alpha = \inf\{x \in \mathbb{R} \mid \mathbb{P}(X \leq x) \geq \alpha\} \tag{5.1}$$

Hence, the mean of X is $\mu(X) = q_{0.5}$, and the IQR is

$$\text{IQR}(X) = q_{0.75} - q_{0.25} \tag{5.2}$$

Value-at-Risk (VaR) is a metric used (mainly in financial risk management) to quantify potential loss over a time horizon with a probability (Acerbi & Tasche, 2002; Holton,

2003). In the context of algorithm performance, it can be described as: *the minimum performance drop threshold over a time horizon T such that the probability of surpassing the threshold is β*. Referring to Figure 5.1, VaR can be expressed as (Acerbi & Tasche, 2002):

$$\begin{aligned} \mathrm{VaR}_\beta(X) &= \inf\{x \in \mathbb{R} \mid \mathbb{P}(X \leq x) \geq \beta\} \quad \text{for } \beta \in (0,1) \\ &= q_{\alpha=\beta} \end{aligned} \tag{5.3}$$

While VaR is the threshold, *Conditional Value-at-Risk* (CVaR) is the expected (average) performance drop below the threshold (Chan et al., 2020), i.e.

$$\mathrm{CVaR}_\beta(X) = \mathbb{E}[X \mid X \leq \mathrm{VaR}_\beta(X)] \tag{5.4}$$

In summary, IQR is the width of performance distribution and CVaR is the average performance drop below a threshold. Often, CVaR is described as the extent of the lower tail of the distribution.

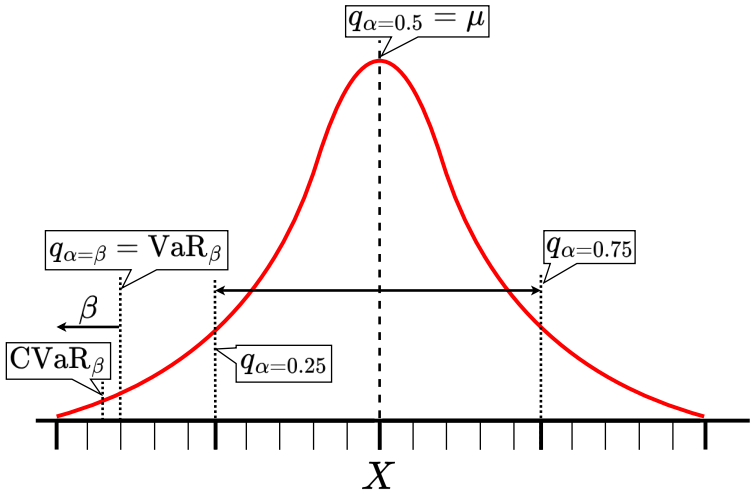


Figure 5.1: *Example of a Gaussian probability density function $X \sim \mathcal{N}(\mu, \sigma^2)$. Note that $q_\alpha \in$ support(X) and $\alpha = \mathbb{P}(X \leq q_\alpha)$*

Metric definitions. IQR and CVaR are used to assess variability across runs and across time (per-run) during training as follows:

1. IQR across time (DT) — captures the degree of noise in the performance within runs. It focuses on the higher-frequency variability in the performance.

2. IQR across runs (DR) — quantifies variability across runs. Typically standard deviation is used for this purpose; however, IQR is more robust since it is less sensitive to outliers.

3. CVaR on Differences across time (SRT) — evaluates short-term performance drops within runs. These can be considered high frequency drops. The random variable in Equation 5.4 is $X_t = R_t - R_{t-1}$, where R_t is the return at an evaluation point t.

4. CVaR on Drawdown across time (LRT) — indicates long-term performance drops over longer timescales. The drawdown at time T is the performance drop relative to the highest peak hitherto, i.e. $X_T = R_T - \max_{t \leq T} R_t$. These drops are of low frequency.

5. CVaR across runs (RR) — measures the expected performance of the worst runs. CVaR is applied to the final performance, i.e. $X = R_{t=T}$, of each training run.

Implementation. We adapt the source code provided by Chan et al. (2020) — found at `https://github.com/google-research/rl-reliability-metrics` — to evaluate these metrics. It should be noted that the metrics *IQR across time* (DT), *IQR across runs* (DR) and *CVaR across runs* (RR) can be determined at several points along the training runs to capture how reliability evolves. To achieve this, the training run is divided into three timeframes, namely: beginning, middle and end. Within each timeframe, the metric values are aggregated into a single value. This significantly reduces noise while also tracking reliability across the run. However, an aggregated performance across these timeframes can also be provide a concise basis for comparison, as detailed in (Chan et al., 2020).

Further, the median results of these metrics (for any set of algorithms) are converted into rankings per-environment. This enables comparison across different environments, which commonly have different ranges of rewards. Thus, the mean ranking across tasks is used to summarise an algorithm's performance across the tasks. Note the following hyperparameters for some metrics:

1. Evaluation frequency — defines the sampling rate at which metrics are evaluated. Apart from *Long-term across Time* (LRT), the rest of the reliability metrics are insensitive to the evaluation frequency. This is because LRT depends explicitly on the number of timepoints the metric is evaluated.

2. Window size — is the width of the sliding window (interval) up on which *Dispersion across Time* (DT) is computed.

3. Frequency threshold — is used for low-pass and high-pass filtering to suppress unwanted frequency components during computation of *Dispersion across Time* (DT) and *Dispersion across Runs* (DR). Note that higher-frequencies are reserved for DT, while lower-frequencies are for DR.

These hyperparameters should be reported along with the metric results, and more details about them can be found in (Chan et al., 2020). Now that the reliability metrics are established, the next section will discuss the evaluation framework for hyperparameter sensitivity.

5.2.2 Hyperparameter Sensitivity Metrics

Suppose the return (cumulative reward) is represented by $R(\omega, e, h, \kappa)$, where $w \in \Omega$ is an algorithm, $e \in \mathcal{E}$ is an environment, $h \in H^{\omega}$ is a hyperparameter setting, and $\kappa \in \mathcal{K} \subset \mathbb{N}$ is a random number seed. Typically during the evaluation of RL algorithms, many training runs are conducted to estimate the expected performance $\hat{R}(\omega, e, h) = \frac{1}{|\mathcal{K}|} \sum_{\kappa \in \mathcal{K}} R(\omega, e, h, \kappa)$. Performing hyperparameter sensitivity analysis requires capturing the expected performance across sets of environments. The reason being to observe the degree in which hyperparameter tuning per-environment impacts performance.

Adkins et al. (2024) defined the hyperparameter sensitivity Φ for a given algorithm as

$$\Phi(\omega) = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \max_{h \in H^{\omega}} \Gamma(\omega, e, h) - \max_{h \in H^{\omega}} \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \Gamma(\omega, e, h) \tag{5.5}$$

where $\Gamma(\omega, e, h)$ is the normalised expected performance called *score*. Hence, Equation 5.5 is the difference between an algorithm's per-environment tuned score and its cross-environment tuned score. The per-environment tuned score is a result of the best hyperparameter tuning per environment, while the cross-environment tuned score is due to the best fixed hyperparameter setting for all the environments. To determine $\Gamma(\omega, e, h)$, the expected performance $\hat{R}(\omega, e, h)$ is normalised via *min-max scaling* (Murphy, 2012) using,

$$\Gamma(\omega, e, h) = \frac{\hat{R}(\omega, e, h) - \min\left[\hat{R}(\omega, e, h)\right]}{\max\left[\hat{R}(\omega, e, h)\right] - \min\left[\hat{R}(\omega, e, h)\right]} \tag{5.6}$$

which linearly maps values to the interval $[0, 1]$, and ensures that scores are commensurable across different environments.

While Equation 5.5 effectively captures the hyperparameter tuning effort, it does not highlight which specific hyperparameter(s) contributed the most to the performance. It may be valuable for RL practitioners to know the key contributing hyperparameter(s), to prioritise it(them) over others. To tackle this, consider a hyperparameter setting to be a tuple $h = \{\zeta_i\}_{i=1}^{I}$ where ζ_i is a fixed value of the i^{th} hyperparameter of the algorithm. During tuning, hyperparameter i takes values from its domain. The algorithm can be conceptualised as model that takes hyperparameter values as inputs and performance as the output. This conceptualisation enables the use of other sensitivity analysis approaches for studying system behaviour (Konakli & Sudret, 2016), such as the *one-factor-at-a-time* (OAT) (Saltelli et al., 2004) and variance-based methods (Opgenoord et al., 2016).

OAT is a simple and widely used method for exploring the model's input space to assess

its effects on the output (Saltelli & Annoni, 2010). It involves sequentially moving one input variable in isolation while keeping others at their nominal values. The limitation of this approach is that it does not fully examine the input space, since it cannot capture interaction effects of the inputs (Christopher & Patil, 2002). However, it is intuitive and can be used as an initial guide to gain understanding of the influence of inputs on the output. In variance-based analysis (Saltelli et al., 2010; Opgenoord et al., 2016), the most influential input variable is determined by decomposing the variance of the output into components attributable to variations in each input. We adapt OAT with variance-based analysis to compute the relative contribution of hyperparameter i to the algorithm's performance variance as

$$S_i^{OAT}(\omega, e) = \frac{\mathrm{Var}_{h_i}\left[\Gamma(\omega, e, h)\right]}{\sum_{j=1}^{I} \mathrm{Var}_{h_j}\left[\Gamma(\omega, e, h)\right]} \tag{5.7}$$

where $\mathrm{Var}_{h_i}[\cdot]$ denotes variance due to varying h_i and $S_i^{OAT}(\omega, e)$ is sensitivity per-environment due to hyperparameter h_i. In computing $S_i^{OAT}(\omega, e)$, we do the following:

1. Vary a single hyperparameter h_i over its range while keeping others fixed.
2. Compute corresponding scores $\Gamma(\omega, e, h)$ over the range of values of h_i.
3. Compute the variance of the scores $\mathrm{Var}_{h_i}\left[\Gamma(\omega, e, h)\right]$ due to varying h_i.
4. Repeat steps (1) - (3) for each hyperparameter and end up with a collection of score variances $\{\mathrm{Var}_{h_i}\left[\Gamma(\omega, e, h)\right]\}_{i=1}^{I}$ for all hyperparameters.
5. Sum the variances $\sum_{i=1}^{I} \mathrm{Var}_{h_i}\left[\Gamma(\omega, e, h)\right]$ to determine the denominator in Equation 5.7.
6. Divide each hyperparameter's variance score $\mathrm{Var}_{h_i}\left[\Gamma(\omega, e, h)\right]$ by the denominator (sum of variances) to determine the relative contribution of each hyperparameter.

Note that S_i^{OAT} resembles *Sobol's indices* (Sobol, 2001), which captures the true contribution of each input to the outcome variability by accounting for interaction effects of the inputs. Nevertheless, these are computationally expensive and mostly require knowledge about the model (Tosin et al., 2020). The mean of S_i^{OAT} across environments can be used to summarise the contributions of the hyperparameters to the algorithm's overall performance. In the next section, we will use the metrics discussed in this section to evaluate RL algorithms.

5.3 Experimental Evaluation

In this section, we evaluate the reliability and hyperparameter sensitivity of RL algorithms in robotic reaching tasks. The tasks encompass arm configurations listed in

Table 5.1 with various reward formulations. One reward formulation is dense-rewards, defined using

$$r = -\omega_1 \|P_{ee} - P_g\|_2^2 - \omega_2 \|action\|_2^2$$

where $[\omega_1, \omega_2] = [1, 1]$ are distance and control weights. P_{ee} and P_g are respectively end-effector and target(goal) positions. Another reward formulation is sparse-rewards, described using

$$r = \begin{cases} 0, & \text{if } \|P_{ee} - P_g\|_2 < 0.05 \text{ meters} \\ -1, & \text{otherwise} \end{cases}$$

where 0.05 meters is the threshold distance between the end-effector and target position. Note that these reward formulations were first introduced in Section 4.5.1 as Equations 4.9 and 4.10, respectively.

In each experiment, both the end-effector of the manipulator and the target are initialised at random locations. This is good training practice that prevents environment overfitting when training RL agents (Whiteson et al., 2011). The type of motion control utilised is torque control. Moreover, it is assumed that the environments are frictionless and have fully observable states. All tasks have a maximum of 50 steps per episode and 500 training episodes.

Table 5.1: *Task Experiment descriptions.*

No.	Arm Configuration	Link Lengths [meters] (base to end-effector)
1	1-Link	1.0
2	2-Link	0.95, 0.70
3	UR10 (3-Link)	0.612, 0.572, 0.116

Note that the UR10 robot (Petrone et al., 2025) is operated on a vertical plane that includes the effects of gravity, and only three arm joints are enabled with the wrist joints disabled. This effectively reduces the system to a *3-Link* manipulator. *1-Link* and *2-Link* manipulators are operated on a horizontal plane to ignore the effects of gravity.

Outline of Results. Section 5.3.1 provides details of the algorithms and methods employed to determine the experimental results. Section 5.3.2 presents performance curves of the algorithms. The results of the reliability analysis are located in Section 5.3.3, while for hyperparameters sensitivity analysis are in Section 5.3.4.

5.3.1 Implementation Specifications

The assessment of reliability and hyperparameters sensitivity provides insight about the algorithm's robustness. Furthermore, to some extent, hyperparameters sensitivity provides some insight about the algorithm's generalisation capabilities. The algorithms employed in our analysis are DDPG and SAC, which are state-of-the-art algorithms suitable for continuous control problems (as discussed in Chapter 2). The algorithms are utilised in their basic form (i.e. unaugmented) for tasks with dense rewards. However, the algorithms extensively fail in tasks with sparse rewards, so we augmented them with Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) to improve their performance.

The implementations of the algorithms consist of both the policy (actor) and value function (critic) networks with 2 hidden layers of 256 units each, and a target update rate of 0.005. The actions are joint torques, while the states include joint positions, joint velocities, end-effector position, end-effector velocity, and the distance of end-effector from the target location. The algorithms were assessed across at most $5,000$ episodes depending on their convergence per-environment. Note that the algorithms fail to find acceptable policies for the *UR10* manipulator task with dense rewards, therefore the task is entirely omitted from the study.

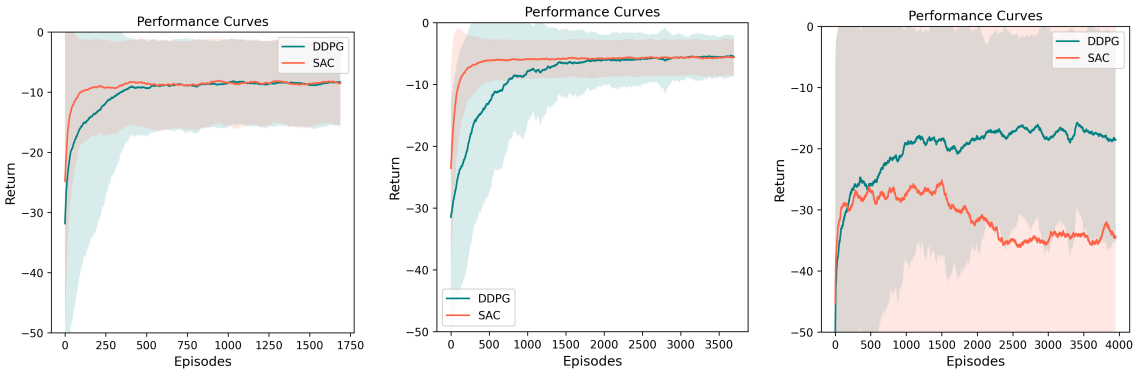
For reliability analysis, the values assigned to parameters during the analysis are 10^4 for window size, 0.01 Hz threshold frequency for low-pass and high-pass filtering, and 5000 timesteps for the evaluation frequency. For hyperparameters sensitivity analysis, five hyperparameters are considered for the DDPG algorithm, while four are considered for the SAC algorithm. The five DDPG hyperparameters are the actor learning rate, critic learning rate, batch size, buffer size and exploration noise.

The actor and critic learning rates dictate the updating speeds of the policy and value function networks, respectively, during training. Exploration noise is the standard deviation value of the Gaussian noise used for action exploration during training. The batch size is the number of training samples utilised per update. Buffer size is the total number of training samples stored in memory (Experience Replay) from which batches are taken from in each update step.

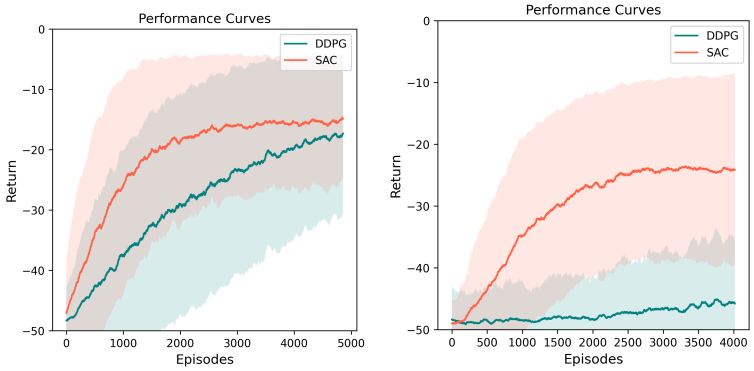
For SAC, the four hyperparameters considered are the temperature coefficient (α), learning rate, batch size and buffer size. The temperature coefficient (α) modulates the exploration behaviour of the algorithm during training. The learning rate controls the updating speeds of both the policy and value function networks. Note that the *Tianshou* (Weng et al., 2022) RL library was employed to facilitate implementation and training of the algorithms, while *PyBullet* (Coumans & Bai, 2016) physics simulator was used to simulate the task environments.

5.3.2 Performance Analysis

The hyperparameter settings with good performance per-environment (or across environments) were identified through hyperparameter sweeping (shown in Table 5.3 and discussed later in Section 5.3.4). The results presented in this section are for hyperparameters tuned for each environment. 20 training runs with different random seeds were carried out in each task per-algorithm. Figure 5.2 shows the performance curves during learning of the tasks in terms of returns (cumulative rewards). Note that direct comparisons of the results in the dense- and sparse-reward settings should be made cautiously and in context. The reason being that the algorithms in the sparse-reward settings were augmented with HER to be able them to solve the tasks.



(a) *1-link dense-reward* **(b)** *1-link sparse-reward* **(c)** *2-link dense-reward*



(d) *2-link sparse-reward* **(e)** *UR10 sparse-reward*

Figure 5.2: *Performance curves during learning of DDPG and SAC algorithms across tasks. The line in high contrast represents the mean, while the shaded region around the line is the standard deviation. Note that the results for UR10 dense-rewards are not shown because both algorithms failed to learn acceptable policies.*

Figure 5.2 clearly depicts SAC learning faster than DDPG in most tasks. This is demonstrated by SAC being first to reach the peak performance in all the tasks except for a single task. In both the *1-link* arm tasks, the algorithms settle at the same performance. This shows that in simpler tasks the algorithms can have indistinguishable final performance. Moreover, we notice the mean peak performance in each task scaling

in accordance with the number of links. It is about -10, -20, -30 for 1-link, 2-link, and UR10 arm tasks, respectively.

This is due to errors at the end-effector scaling with link lengths and number of joints (as discussed in Section 4.5.2). As the lengths of the links of a manipulator and its number of joints are increased, errors in the joint angles are amplified at the end-effector as illustrated by Equation 4.11, reproduced below.

$$\|\delta x\|_2 = \left\| \sum_{k=1}^{n} J_k(\theta)\delta\theta_k \right\|_2 \leq \epsilon \sum_{i=1}^{n} i l_i$$

where δx is error at the end-effector, $\delta\theta_k$ is angle error of the k^{th} joint, and $J_k(\theta)$ is the k^{th} column of the Jacobian matrix. n is the number of DoF/joints, while l_i is the i^{th} link on the arm. ϵ is the worst-case joint error, i.e. $|\delta\theta_k| \leq \epsilon$ for $k = 1, \cdots, n$ (see Appendix C.1 for details on Equation 4.11).

The increasing error rate at the end-effector makes it more challenging for the agent to meet the threshold distance (< 0.05 m) when dealing with more joints and longer link lengths. As a consequence, the performance of the agents drops, reflecting task hardness, as links become longer and more joints are added. This is further supported by literature (Wang et al., 2020; Edmondson & Petrick, 2025), that shows that tasks which are highly prone to errors present greater learning challenges for RL algorithms, thus resulting in reduced rewards or longer learning time. In the case of *UR10* arm, the presence of gravity introduces further errors and instabilities in the system (Arimoto, 1986; Perrusquía et al., 2020) that increase the hardness of the task and exhibit lower agent performance.

The increasing error rate is also captured by the growing performance variance across tasks with the same reward formulation. For example, by looking at Figures 5.2 (b), (d) and (e) or Figures 5.2 (a) and (c), it is evident that the performance variance worsens as number of joints and lengths of links increase. In a different regard, performance variance in dense-reward settings appears higher than in sparse-reward settings for the corresponding manipulator tasks. This ensues even though the converse is expected since tasks with sparse rewards are more difficult. However, this stems from augmenting the algorithms with HER in the sparse-reward settings. The augmentation improves performance and also avoids the effects of reward shaping.

Reward shaping can introduce inaccuracies in the system which can be exacerbated by the task complexity. To illustrate, *2-link* arm task with dense rewards (Figure 5.2 (c)) has the worst performance variance across all the tasks . In fact, during demonstrations, the arm had slight quivering motion called *chattering*. Since reward shaping is prone to misalignments with the true task objective (Vamplew et al., 2022; Ibrahim et al., 2024), it can lead to unintended agent behaviours (Ibrahim et al., 2024). We speculate that such deviations from the intended policy engendered chattering which contributed

to behavioural variability. Additionally, this combined with the error rate (due to increasing the number of joints and link lengths) worsened the performance variance.

In the *UR10* arm task (Figure 5.2 (e)), DDPG fails to achieve the peak performance. The failure of DDPG can be attributed to its:

1. inadequate exploration to thrive in unstable environments (Henderson et al., 2018).

2. propensity to become stuck in suboptimal policies when high-reward actions are not encountered in the initial training phases (Matheron et al., 2019).

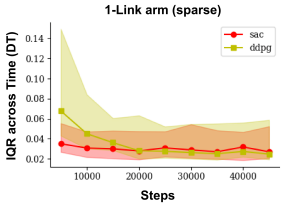
The presence of gravity in this task makes the environment unstable (Arimoto, 1986; Perrusquía et al., 2020), as the arm requires compensatory joint torques to remain upright. In these types of environments, DDPG has been reported to struggle to learn a proper Q-value function estimation since many exploratory paths result in failure (Henderson et al., 2018). Moreover, sparse-reward settings have been found to be responsible for causing DDPG to converge prematurely (Matheron et al., 2019).

5.3.3 Reliability Analysis

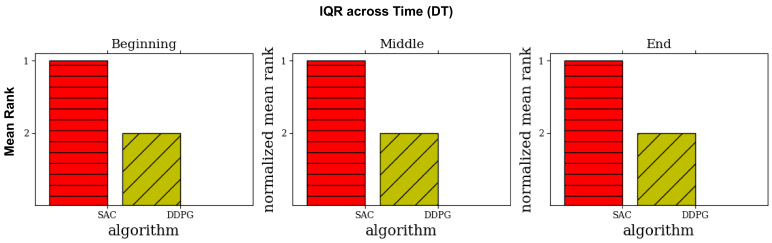
The same experimental setup of hyperparameter settings and number of training runs used in the performance analysis (Section 5.3.2) was employed for reliability analysis in this section. The reliability results of the algorithms in the *1-link* arm task (with sparse rewards) and UR10 arm task are presented in Figures 5.3, 5.4 and 5.5. IQR signifies the performance variability, while CVaR denotes the expected performance value of the worst-performing run (or the worst performance drop within a run) with 5% probability of occurrence. In the figures, relative rankings are the focus and the metric values on the first columns are included to assist interpretation.

Figure 5.3 (a - b) exhibits that SAC has lower degree of variability in the performance within runs than DDPG, across all the timeframes. For variability across runs (Figure 5.3 (c - d)), SAC still outperforms DDPG. DDPG seems to experience slightly more variability at the *beginning timeframe* than the rest. The results corroborate with the standard deviations and fluctuations in the mean performances of the algorithms showcased in Figure 5.2 (b). In Figure 5.3 (e - f), DDPG is at a higher risk of worst performance across runs than SAC, except at the *end timeframe*. SAC is at a lower risk of both short-term and long-term performance drops as highlighted by Figures 5.4 (a - b) and 5.4 (c - d). Note that the differences in the all these outcomes are not statistically significant. However, Figure 5.5 shows the scenarios when the outcomes are statistically significant.

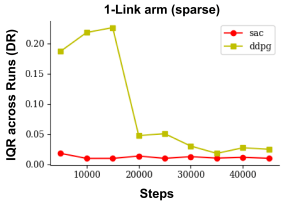
Table 5.2 summarises the reliability metrics ranking across all the tasks, with corresponding plots in Appendix D. For DT, DR and RR, the results reported in the table are for the *end* timeframe. The timeframe is the most critical for comparison, since it signifies the convergence phase of the algorithms' learning. By aggregating the rankings



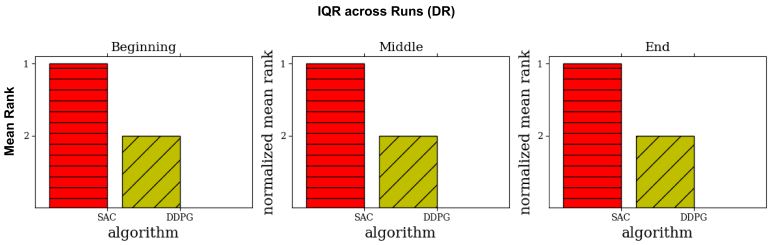
(a) *Dispersion across Time (Raw)*



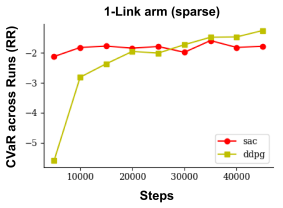
(b) *Dispersion across Time (Ranking per timeframe)*



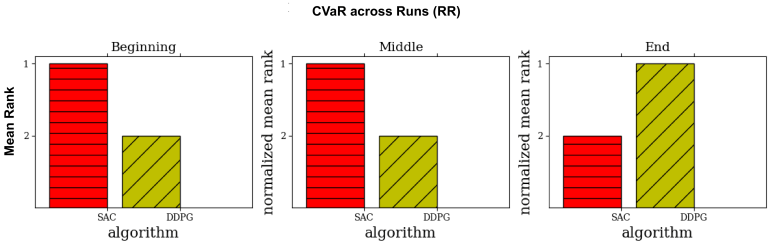
(c) *Dispersion across Runs (Raw) - for a representative run*



(d) *Dispersion across Runs (Ranking per timeframe)*



(e) *Risk across Runs (Raw) - for a representative run*



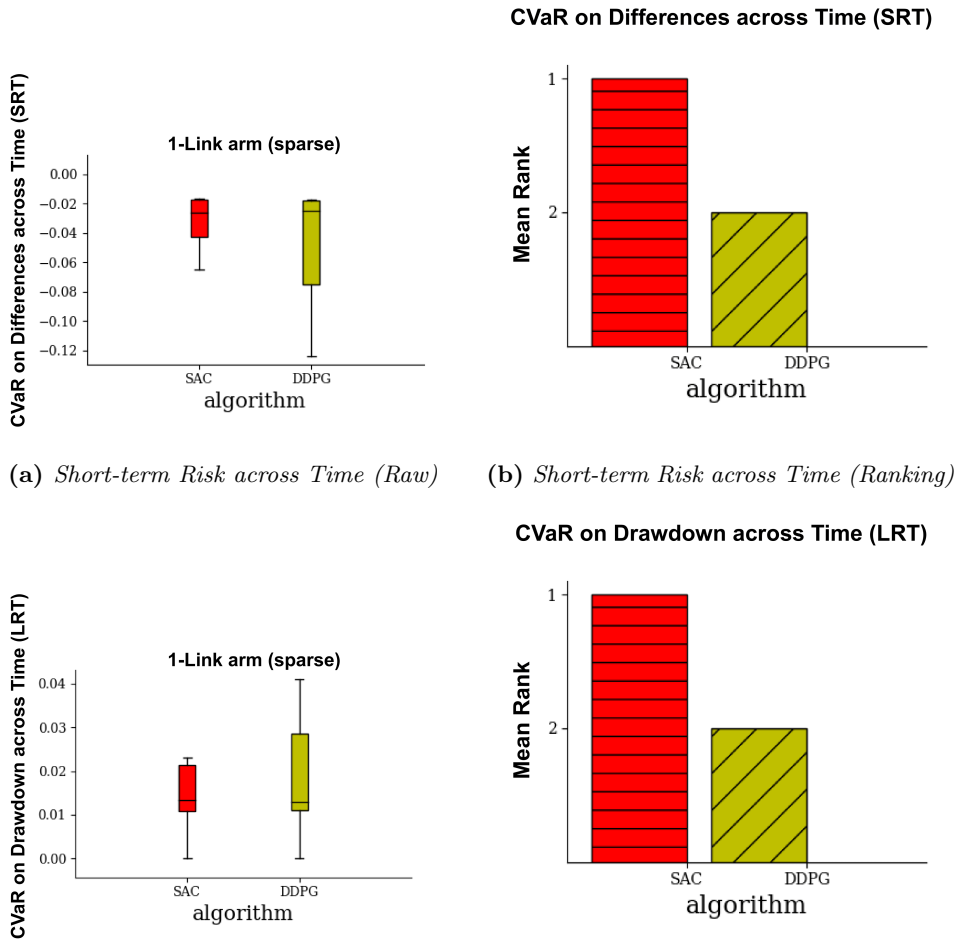
(f) *Risk across Runs (Ranking per timeframe)*

Figure 5.3: *Reliability metrics rankings for the 1-link arm task with sparse rewards during training. Note that rank 1 always indicates better metric outcome. (a), (c) and (e) respectively illustrate raw metric values of dispersion across time (DT), dispersion across runs (DR), and risk of worst performance (with 5% probability) across runs (RR). (b), (d) and (f) are corresponding metric rankings per timeframe during training. Note that less positive values in raw metrics represent better outcome. Additionally, statistically significant differences in ranking should have a black horizontal line above the bars, its absence indicates non-significance.*

across the tasks, SAC emerges as the overall more reliable algorithm.

5.3.4 Hyperparameter Sensitivity Analysis

To explore the hyperparameters space during evaluations, a single hyperparameter at a time was varied from its nominal value while keeping others frozen. The nominal values of the hyperparameter setting were determined through sweeping the search space as shown in Table 5.3. Table 5.4 outlines the results of the algorithms' hyperparameter



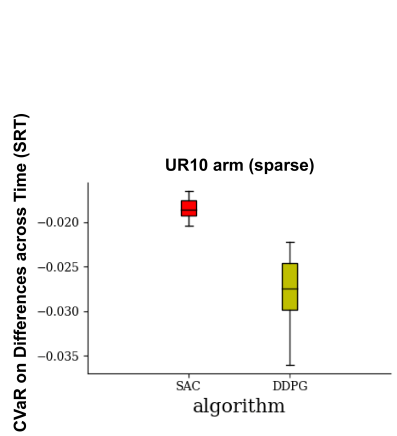
(a) *Short-term Risk across Time (Raw)* **(b)** *Short-term Risk across Time (Ranking)*

(c) *Long-term Risk across Time (Raw)* **(d)** *Long-term Risk across Time (Ranking)*

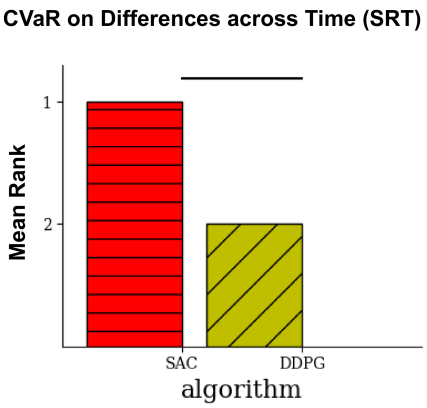
Figure 5.4: *Reliability metrics ranking for the **1-link arm task with sparse rewards** during training. Rank 1 indicates better metric outcome. (a) shows box plots of risk of performance drop with 5% probability over short-terms during the runs (SRT), while (b) depicts the corresponding mean rankings. (c) portrays risk of performance drop with 5% probability across long-terms during the runs (LRT), while (d) outlines the corresponding mean rankings. Note that more positive values in (a) denote better risk, and in (c) less positive values are superior. Additionally, statistically significant differences in ranking should have a black horizontal line above the bars, its absence indicates non-significance.*

sensitivity Φ along with the contribution of each hyperparameter to the performance variability across the tasks $\hat{S}_i^{OAT}$.

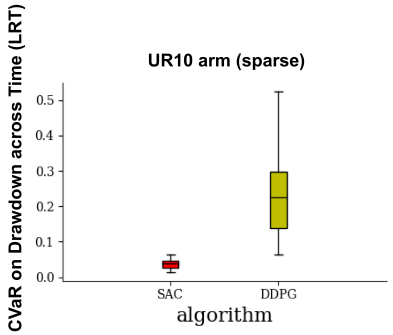
To determine Φ, the best performing hyperparameter settings per environment and across environments were employed as prescribed by Equation 5.5. For computing the variance contributions $\hat{S}_i^{OAT}$, hyperparameters were varied from their nominal, per-environment tuned values. In each hyperparameter setting, 20 training runs with



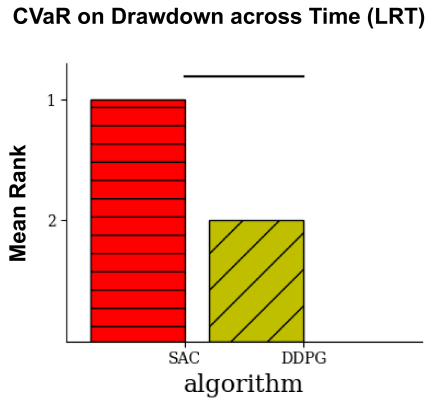
(a) *Short-term Risk across Time (Raw)*



(b) *Short-term Risk across Time (Ranking)*



(c) *Long-term Risk across Time (Raw)*



(d) *Long-term Risk across Time (Ranking)*

Figure 5.5: *Reliability metrics ranking for the* ***UR10 arm task with sparse rewards*** *during training. Rank 1 indicates better metric outcome. (a) shows box plots of risk of performance drop with* 5% *probability over short-terms during the runs (SRT), while (b) depicts the corresponding mean rankings. (c) portrays risk of performance drop with* 5% *probability across long-terms during the runs (LRT), while (d) outlines the corresponding mean rankings. Note that more positive values in (a) denote better risk, and in (c) less positive values are superior. Additionally, statistically significant differences in ranking should have a black horizontal line above the bars, its absence indicates non-significance.*

different random seeds were conducted. The variance of the median performance scores across the runs were utilised in Equation 5.7 to determine $\hat{S}_i^{OAT}$. Figures in Appendix E demonstrate the variation of hyperparameters for each algorithm per-environment.

DDPG is about three times more sensitive to its hyperparameters than SAC across the tasks — as revealed by values of Φ in Table 5.4. The algorithm required more tuning effort, and consequently displayed poorer stability and algorithm-level generalisation

Table 5.2: *Summary of reliability ranking results across the tasks. The algorithm with a higher rank between DDPG and SAC is listed under the metric. Note that DT = IQR across time, DR = IQR across runs, RR = CVaR across runs, SRT = CVaR on Differences across time, and LRT = CVaR on Drawdown across time. Moreover, the star ($\star$) denotes the algorithm outranks its counterpart with statistical significance.*

Rewards Setting	Arm	DT	DR	RR	SRT	LRT
Dense	1-link	DDPG	SAC	DDPG	DDPG	DDPG*
	2-link	SAC	DDPG	DDPG	SAC	SAC*
Sparse	1-link	SAC	SAC	SAC	SAC	SAC
	2-link	SAC	SAC	SAC	SAC*	SAC
	UR10	DDPG*	DDPG	SAC	SAC*	SAC*

Table 5.3: *Hyperparameter search space for DDPG and SAC algorithms.*

Algorithm	Hyperparameter	Search min	Search max	Nominal
DDPG	Batch Size	32	512	128
	Buffer Size	10^3	10^7	10^5
	Exploration Noise	0.01	1.0	0.1
	actor learning rate	10^{-5}	10^{-1}	10^{-3}
	critic learning rate	10^{-5}	10^{-1}	10^{-3}
SAC	α coefficient	0.05	1.0	0.2
	Batch Size	16	256	64
	Buffer Size	10^3	10^7	10^5
	learning rate	10^{-5}	10^{-1}	10^{-3}

than SAC. For DDPG hyperparameters, the exploration noise and actor learning rate (for the policy network) contributed about 70% to the algorithm's performance. While for SAC hyperparameters, the α coefficient and learning rate (for both the policy and Q-function networks) contributed about 90%. In both algorithms, the exploration-related and policy-model learning hyperparameters strongly influence performance, which aligns with literature (Sutton & Barto, 2018).

5.4 Discussion

The evaluation of performance, reliability and hyperparameter sensitivity of DDPG and SAC was performed across multiple reaching tasks. The reaching tasks consisted of *1-link* arm, *2-link* arm and *UR10* arm with both dense and sparse rewards, expect for the *UR10* arm with solely sparse rewards. The following observations can be made about the results:

Performance analysis results. The results in the performance analysis indicate that

Table 5.4: *Hyperparameter sensitivity of algorithms across tasks.*

Algorithm	Φ	Hyperparameter	$\hat{S}_i^{OAT}$
DDPG	0.1702	Batch Size	0.1199
		Buffer Size	0.0442
		Exploration Noise	0.2236
		actor learning rate	0.4712
		critic learning rate	0.1411
SAC	0.0513	α coefficient	0.1843
		Batch Size	0.0320
		Buffer Size	0.0611
		learning rate	0.7226

SAC generally learns faster than DDPG, however in most cases they reach equivalent peak performance. While the use of reward shaping simplifies learning, it introduces unwanted chattering in the robot motion. In most tasks, DDPG and SAC converge to policies with equivalent returns. Without other performance metrics to consider, we would be forced to conclude that DDPG and SAC have similar performance.

Reliability analysis results. The results in the reliability analysis reveal SAC to be a more reliable algorithm compared to DDPG. It should be noted that the difference in reliability in most cases is not statistically significant. Generally, DDPG has a higher risk of performance drop or worst performance across the tasks during training. This underscores its likelihood of performance degradation during training as denoted in (Duan et al., 2016; Haarnoja et al., 2018).

Hyperparameter sensitivity analysis results. DDPG is more sensitive than SAC. This corroborates with Haarnoja et al. (2018), who asserted that SAC is less sensitive than DDPG although did not provide empirical evidence. This means SAC is a more stable algorithm than DDPG. From the selected set of hyperparameters used for tuning, both algorithms' performance were significantly affected by hyperparameters for exploration and policy-model learning. The policy model aims to approximate the optimal policy, while exploration facilitates its discovery. Hence these results are consistent with intuition.

5.5 Concluding Remarks.

This chapter successfully introduces and evaluates frameworks for assessing the robustness of RL algorithms, specifically reliability and hyperparameter sensitivity of the algorithms. We also utilise cumulative rewards in conjunction with agent demonstrations to evaluate long-term consequences (behaviour) of the algorithms as suggested by Henderson et al. (2018). At this stage, we have all the necessary tools to assess the *efficiency of RL agents* (Osband et al., 2019) based on robustness, exploration and long-term consequences. In the next chapter, we deploy the evaluation frameworks

on a more intricate high-dimensional manipulation task. This allows us to assess the utility of the *multi-faceted performance metric* in comparing RL algorithms.

Chapter 6

Multi-dimensional Performance Metric

6.1 Introduction

The field of RL continues to introduce novel algorithms at a rapid pace (Garnier et al., 2021; Sinha & Pradyumna, 2022; Terven, 2025), with the intention of addressing various learning capabilities including, but not limited to, exploration (Ladosz et al., 2022), generalisation (Kirk et al., 2023), robustness (Moos et al., 2022) and reliable agent behaviour (Chan et al., 2020). For example, UCRL2 was designed to provide provably near-optimal exploration in tabular settings (Jaksch et al., 2010), while SAC was developed to enhance robustness (stability) and sample efficiency in non-tabular domains (Haarnoja et al., 2018). Nevertheless, the capabilities of new algorithms are typically examined solely on cumulative reward, overlooking other meaningful performance metrics.

As established in Section 1.3, cumulative rewards assess the integrated general learning abilities of algorithms without necessarily isolating core capabilities (Osband et al., 2019). In contrast, most real-world scenarios involve making trade-offs with respect to different performance metrics (Xu et al., 2020). For instance, a practitioner might want to select an algorithm that best explores the state-action space but with less generalisability or slightly inferior robustness, or vice-versa. Using an aggregate metric, such as cumulative reward, can make this challenging to conduct in a principled manner. Furthermore, the proclivity of cumulative rewards to misrepresent desired agent behaviour (Andrychowicz et al., 2017; Henderson et al., 2018; Ibrahim et al., 2024) can hinder effective diagnosis of deficiencies in specific capabilities of algorithms (Osband et al., 2019). For example, an algorithm may attain high cumulative rewards yet exhibit a bottleneck in robustness, thus requiring considerable hyperparameter tuning to ensure reliable performance.

Furthermore, relying entirely on cumulative rewards to compare algorithms can obscure

the underlying reasons for performance differences (or similarities) due to *value ambiguity* (Yau, 2024). This phenomenon arises because cumulative rewards capture the overall quality of a policy without revealing how or why it selects actions (i.e. behaves), resulting in comparisons that may be misleading or difficult to interpret. For example, algorithms may attain similar cumulative rewards, but the behaviours of the policies they converge to can differ substantially. This is actually observed in our experiments where learned policies of SAC produce smoother robot motion than TD3, yet they have almost the same mean cumulative reward. To address this limitation, we studied metrics in the previous chapters that capture different capabilities of algorithms, with the goal of developing a multi-dimensional performance metric.

Since an *efficient RL agent* concurrently addresses challenges in *exploration*, *generalisation* and *long-term consequences* (Osband et al., 2019), in this chapter we propose a multi-faceted performance metric over these dimensions. In the previous chapters, we

1. Introduced metrics for *exploration*, i.e. *Effort of Sequential Learning* (ESL) and *Optimal Movement Ratio* (OMR) (Nkhumise et al., 2025).

2. Studied metrics for *robustness*, mainly focused on reliability and hyperparameter sensitivity. This is because robustness is a necessary condition for generalisation (Xu & Mannor, 2012) and is easier to evaluate directly than generalisation (Kirk et al., 2023).

3. Discussed how *agent demonstrations* are necessary to support cumulative rewards (Henderson et al., 2018), when examining *long-term consequences (behaviour)* of RL algorithms.

The aim of this chapter is to integrate these various metrics into an evaluation framework and demonstrate its utility in complex environments. We define a complex environment as one with high-dimensional continuous state and action spaces, as commonly found in real-world problems (Dulac-Arnold et al., 2019; Tang et al., 2025). Specifically, we employ a 7 degrees-of-freedom (DoF) robotic arm simulated in MuJoCo (Todorov et al., 2012). Our contributions include (1) extending exploration metrics developed in Chapter 3 (ESL and OMR) to high-dimensional continuous settings and (2) demonstrating the use of a multi-faceted performance metric in RL based on exploration, robustness and long-term behaviour (consequences).

6.2 Evaluation Framework

In this section, we provide a brief recap of evaluation metrics for exploration, robustness and long-term consequences (behaviour) — as discussed in the previous chapters. This is followed by detailed accounts on extending some of these metrics to high-dimensional continuous domains to alleviate computational complexity.

Exploration. We introduced *Effort of Sequential Learning* (ESL) and *Optimal Movement Ratio* (OMR) to capture the exploratory processes of algorithms. These are

based on the travelled path lengths in the occupancy measure space by algorithms during training. ESL quantifies the cost of exploration with respect to the direct path between the initial policy and the final (optimal) policy. OMR measures the fraction of movements in the space that directly reduce the distance between the initial and final policies. Moreover, we can visualise the policy evolution during training. This framework can elucidate the effectiveness of the algorithm's exploration strategy in finding the optimal policy during training.

Robustness. We evaluated the robustness of algorithms via computing their reliability and hyperparameter sensitivity. To capture reliability, we employed Inter-quartile range (IQR) and conditional value at risk (CVaR) across time and runs to examine the performance dispersion (variability) and risk of performance drop of algorithms as proposed by Chan et al. (2020). To measure hyperparameter sensitivity, a metric developed by Adkins et al. (2024) (Equation 5.5) was adapted. These metrics illustrate how easy it is to apply an algorithm to new problems, and how consistent and stable the algorithm is.

Long-term consequences (behaviour). We highlighted that the expected returns (mean cumulative rewards) should be interpreted in conjunction with observations of the agent behaviour during the evaluation phase. Both the learning curves and behavioural descriptions of the agent should be reported when presenting results. This is because the expected returns can mischaracterise the quality or desirability of agent behaviour (Henderson et al., 2018; du Preez-Wilkinson & Gallagher, 2020; Yau, 2024), particularly in environments where rewards are not innate (Andrychowicz et al., 2017). For instance, in robotic control, the motion quality — such as smoothness — can be specified alongside learning curves to comprehensively characterise the (long-term) behaviour of the agent.

While these metrics represent interesting aspects of algorithms, they can be computationally expensive or intractable when applied to high-dimensional continuous problems, such as robotics. This is mainly due to the large dimensionality of the state-action space. With the exception of the *long-term behaviour* metric, in the following subsections, we present adaptations of *exploration* and *robustness* metrics to high-dimensional continuous domains. Finally, we discuss how to effectively consolidate and present all the aforementioned metrics.

6.2.1 Exploration

We showed in Chapter 5 that the occupancy measure of a policy for episodic environments is given by Equation 3.34 as $v_\pi^H(s,a) = \mathbb{P}(s,a \mid \pi,\mu)$, where $\mathbb{P}(s,a \mid \pi,\mu)$ is the joint probability distribution of state-action pairs encountered when deploying π in the environment. We estimate $\mathbb{P}(s,a \mid \pi,\mu)$ from a policy dataset of state-action pairs $\mathcal{D}_\pi = \{(s_{(i)}, a_{(i)})\}_{i=1}^m$ collected during the rollout of π. This is because $\mathcal{D}_\pi = \{(s_{(i)}, a_{(i)})\}_{i=1}^m \sim v_\pi^H(s,a)$ as previously discussed in Section 3.5. We then denote the distance between π and π' in the policy space as the Wasserstein dis-

tance $\mathcal{W}_p^p(v_\pi^H, v_{\pi'}^H)$. The Wasserstein distance $\mathcal{W}_p^p(v_\pi^H, v_{\pi'}^H)$ is computed using the policy datasets via OTDD $d_{OT}(\mathcal{D}_\pi, \mathcal{D}_{\pi'})$ as described in Section 3.5.1. This is a nested process that requires mapping actions to distributions over the state space $\mathcal{P}(\mathcal{S})$ as $a \to \beta_a(S) \triangleq P_\pi(S \mid A = a),\ a' \to \beta_{a'}(S) \triangleq P_{\pi'}(S \mid A = a') \in \mathcal{P}(\mathcal{S})$ in order to express $\mathcal{W}_p^p(v_\pi^H, v_{\pi'}^H)$ as

$$\mathcal{W}_p^p(v_\pi^H, v_{\pi'}^H) = \min_{\pi \in \Pi(v_\pi^H, v_{\pi'}^H)} \int_{\mathcal{SA}\times\mathcal{SA}} \left(d_{\mathcal{S}}(s, s')^p + \mathcal{W}_p^p(\beta_a, \beta_{a'})\right)\, d\pi \tag{6.1}$$

where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively. Equation 6.1 can then be utilised to compute ESL and OMR. Note that Equation 6.1 has the inner Wasserstein distance $\mathcal{W}_p^p(\beta_a, \beta_{a'})$ and the outer Wasserstein distance $\mathcal{W}_p^p(v_\pi^H, v_{\pi'}^H)$, hence OTDD is described as a *nested process* of computing the distance between $\mathcal{D}_\pi$ and $\mathcal{D}_{\pi'}$ (refer to Section 3.5.1 for more details).

In continuous domains, an obvious approach to adapting ESL and OMR is to simply discretise the states and actions. While this can work for low-dimensional settings like the *Mountain Car* (Moore, 1990; Brockman et al., 2016), it fails in high-dimensional domains due to the curse of dimensionality. That is, the number of state-action pairs required to approximate the distribution to the same level of accuracy grows exponentially with the degrees of freedom (DoF) (Nikovski, 1998; Kober et al., 2013; Gorodetsky et al., 2018).

This presents two main computational challenges, first is estimating the conditional distributions $\beta_a(S) = P(S \mid A = a)$ given $\mathcal{D}_\pi = \{(s_{(i)}, a_{(i)})\}_{i=1}^m$ where $s \in \mathcal{S} \subseteq \mathbb{R}^k$ and $a \in \mathcal{A} \subseteq \mathbb{R}^l$ for $k + l \gg 1$. Since the number of action bins increases exponentially with the number of action dimensions (Lillicrap et al., 2016), estimating $\beta_a(S)$ via binning requires an exponentially growing number of samples. This necessitates the use of a powerful density estimator (e.g. model-based) to approximate $\beta_a(S)$, whereas in Chapter 3 we did not rely on a density model. The second challenge is computing outputs of Equation 6.1. We tackle these challenges as follows:

Estimating the conditional distributions. Unfortunately, kernel density estimation (KDE) fails to estimate the conditional distributions $\beta_a(S)$ for $k + l \gg 1$ due to the curse of dimensionality (Hastie et al., 2009). Alternatives such as Gaussian Mixture Model (GMM) (Reynolds, 2009) and Deep Normalising Flow (Papamakarios et al., 2021) are prone to numerical instabilities that result in sampling extremely large values which were not part of the training dataset (Gepperth & Pfülb, 2021; Köhler et al., 2021). We therefore employ conditional variational autoencoders (cVAEs) (Sohn et al., 2015). These are powerful probabilistic generative models that can model high-dimensional distributions conditioned on given input observations — as discussed in Chapter 2 (Section 2.3.1). They have been successfully applied in large-language models (Ruan & Ling, 2021) and image reconstruction (Zhang et al., 2021a) amongst others. We train cVAE models for each $\mathcal{D}_\pi$ to approximate $\beta_a(S)$ and then produce samples

from this approximated distribution. Note that to compute the inner Wasserstein distance $\mathcal{W}_p^p(\beta_a, \beta_{a'})$ in Equation 6.1, we need to sample states $s_1^{(a)}, \ldots, s_m^{(a)} \sim \beta_a(S)$ and $s_1^{(a')}, \ldots, s_m^{(a')} \sim \beta_{a'}(S)$, respectively.

Computing Wasserstein distances. To compute the Wasserstein distance between any arbitrary empirical distributions $\hat{P}_A$ and $\hat{P}_B$ based on samples $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively, we employ Algorithm 3 based on the work by Sommerfeld et al. (2019). To compute Equation 6.1, we simply replace $\mathcal{W}_p(\hat{P}_A, \hat{P}_B)$ in Algorithm 3 with Sliced

Algorithm 3: Statistical approximation of $\mathcal{W}_p(P_A, P_B)$

Input: Datasets $\mathcal{D}_A, \mathcal{D}_B$, sample size N and number of repetitions B.

for $i = 1, 2, \ldots, B$ **do**

- Sample i.i.d. $X_1, \ldots, X_N \sim \mathcal{D}_A$ and independently $Y_1, \ldots, Y_N \sim \mathcal{D}_B$;
- $\hat{P}_A \leftarrow \frac{\#\{k\,:\,X_k = x\}}{N}$ for all $x \in \mathcal{X}$;
- $\hat{P}_B \leftarrow \frac{\#\{k\,:\,Y_k = y\}}{N}$ for all $y \in \mathcal{Y}$;
- Compute $\hat{\mathcal{W}}^{(i)} \leftarrow \mathcal{W}_p(\hat{P}_A, \hat{P}_B)$;

Output: $\hat{\mathcal{W}}_p^{(N)}(P_A, P_B) \leftarrow \frac{1}{B}\sum_{i=1}^{B} \hat{\mathcal{W}}^{(i)}$

Wasserstein distance (Kolouri et al., 2019; Nguyen, 2025) for the inner Wasserstein distance $\mathcal{W}_p^p(\beta_a, \beta_{a'})$ and use exact Wasserstein distance for the outer Wasserstein distance $\mathcal{W}_p^p(v_\pi^H, v_{\pi'}^H)$. We utilise Sliced Wasserstein distance to contribute to reducing the cost of computing the inner Wasserstein distance, further it can be augmented with Sinkhorn regularisation (Cuturi, 2013). These methods are desirable since they have guarantees, while Algorithm 3 has expected error bounds (Sommerfeld et al., 2019).

6.2.2 Robustness

While reliability metrics are comprehensively evaluated per environment, the hyperparameter sensitivity metric Φ (described by Equation 5.5) requires evaluation across environments. In fact, Φ is dependent on the selection of the set of environments evaluated and the level of granularity with which hyperparameter sweeps are carried out (Adkins et al., 2024). This can be computationally expensive when the environments in the set have high-dimensional continuous state-action spaces. For example, if environments consist of a robotic arm that performs distinct operations, such as *reach, push, pick-and-place*, etc.; it takes up to $100,000$ training time-steps to solve the *reach* task, 1 million to solve the *push* task, and 5 million to solve the *pick-and-place* task. In this instance, a single hyperparameter setting from the sweep can take at least 6.1 million training time-steps. If the number of evaluated hyperparameters is n, each with m possible values, then the total training time-steps across the sweep grows exponentially as $m^n \times 6.1$ million.

To mitigate this, we reduce n and m in conjunction with dealing with a single task environment, e.g. *reach* task, while disregarding others. We propose using proxy

environments, which are cheaper versions of the selected task with distinct *planning horizons* (i.e. maximum number of time-steps per episode). For example, a set of four proxy environments may entail horizon lengths: $\{50, 75, 100, 125\}$. This reduces the total training time-steps across the hyperparameter sweep to $m^n \times 4 \times 100$ thousand — which is about 15 times smaller than the former. However, this assumes that the horizon length H meaningfully impacts the algorithm performance, and that the results would somehow generalise to distinct tasks.

We can verify that the first assumption is satisfied by assessing the effects of horizon length H on task complexity. As discussed in Chapter 4, performance of algorithms generally diminishes with increasing task difficulty. So, by showing that task complexity increases with the horizon length H, we establish that H meaningfully impacts algorithm performance. We illustrate this as follows:

Given a sequential decision-making problem, at each time-step the agent can select from $|\mathcal{A}|$ possible actions. Thus, the total number feasible action sequences over an entire horizon of length H is $|\mathcal{A}|^H$, i.e. exponentially scaling as H increases. This leads to an exponential growth in the search space for optimal policies (Kakade, 2003; Myers et al., 2025). The exponential growth makes exploration and temporal credit assignment more difficult (Jiang & Agarwal, 2018), and significantly impacts policy learning and value estimations especially in stochastic environments (Li et al., 2022).

For instance, Kantas (2009) showed that the optimal policy becomes more complex as H enlarges, requiring more expressive functions to capture the long-term dependencies between states and actions (Park et al., 2025). Li et al. (2022) and Park et al. (2025) demonstrated that the bias and variance in value estimations are exacerbated as H grows — an effect called the *curse of horizon* (Liu et al., 2018). Therefore, task difficulty scales with the horizon length H, and we can conclude that the performance of algorithms is sensitive to changes in H. Additionally, studies such as (Liu et al., 2018; Kallus & Uehara, 2022; Pignatelli et al., 2023) have shown that algorithm performance declines as H increases.

Note that we can think of proxy environments (i.e. environments with varying horizon lengths) as structurally homogenous tasks, in contrast to less structurally homogeneous tasks such as *reach, push, pick-and-place*, etc. Homogeneous tasks can be an essential first stage for assessing within-domain hyperparameter sensitivity and hence robustness. Algorithms exhibiting poor robustness within-domain are unlikely to generalise effectively across domains (Freiesleben & Grote, 2023).

6.3 Experimental Evaluation

In this section, we present experimental evaluations of the multi-faceted performance metric. We begin with describing the environment upon which we conduct the experiments, and subsequently present results.

6.3.1 Experiment Setup

In simulation, we employ a manipulation environment called *Fetch*, introduced by Plappert et al. (2018) — with its python-based software implemented by de Lazcano et al. (2024). The environment decently resemble real-world conditions with moderately high fidelity. It entails a *7-DoF Fetch Robotic* arm with two-fingered parallel gripper. The robot is simulated on MuJoCo and has been used in a few studies about RL algorithms including Andrychowicz et al. (2017). We consider a *reach* task (FetchReach-v4), where a small red spherical ball (see Figure 6.1) is placed randomly in the workspace of the robot at the start of every episode, and the aim is for RL agent to move the end-effector to the spherical ball.

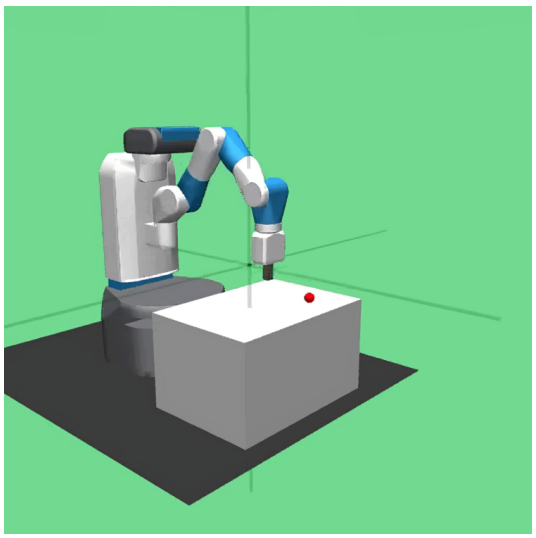


Figure 6.1: *Illustration of the manipulation task (FetchReach-v4) considered for experimental evaluation of RL algorithms. It consist of a 7-DOF robotic arm with two-fingered parallel gripper.*

The environment's reward settings are sparse, with $r = 0$ when the end-effector coincides with the spherical ball, otherwise $r = -1$ at each step. The default task has a horizon length of 100 (i.e. maximum number of steps per episode) and 1000 training episodes. For evaluating hyperparameter sensitivity metric, the variant horizon lengths utilised are $\{25, 50, 75, 100, 125\}$. The algorithms deployed are DDPG, SAC and TD3 with Hindsight Experience Reply (HER) augmentation (Andrychowicz et al., 2017) to enable them to solve the task. These algorithms were selected because they are state-of-the-art algorithms suitable for continuous control problems (as discussed in Chapter 2).

Our experiments aim to address the following questions:
1. *Can a multi-dimensional performance metric provide richer and more interpretable results than a baseline learning curve?*
2. *Which algorithm is better suited for reaching tasks using a 7-DoF Fetch Robotic arm?*

Outline of Results. Section 6.3.2 details the implementation underlying the experimental results. Performance (learning) curves of the algorithms are provided in Section 6.3.3, followed by results of the reliability analysis in Section 6.3.4, hyperparameter sensitivity analysis in Section 6.3.5, and exploration analysis in Section 6.3.6. Finally, the results are consolidated in Section 6.3.7.

6.3.2 Implementation Specifications

The implementations of the algorithms consist of both the policy (actor) and value function (critic) networks with 3 hidden layers of 256 units each, and a target update rate of 0.05. A discount factor $\gamma = 0.99$, batch size of 256 and experience replay buffer of 10^6 were used for training of the algorithms. ADAM optimiser (Kingma & Ba, 2017) was utilised in the neural network models. During evaluation of the exploration metrics, we used $L2$ distance as metric for the state space and collected policy datasets $\mathcal{D}_\pi$ after every 50 updates of the policy model. Note that the *Tianshou* (Weng et al., 2022) RL library was employed to facilitate implementation and training of the RL algorithms.

For reliability analysis, the values assigned to parameters during the analysis are 10^4 for window size, 0.01 Hz threshold frequency for low-pass and high-pass filtering, and 5000 timesteps for the evaluation frequency. For hyperparameter sensitivity analysis, we only focused on capturing the hyperparameter tuning effort using Equation 5.5. Instead of determining the relative contribution of each hyperparameter to algorithm performance (as we did in Chapter 5), we assumed the most influential ones are the network *learning rates* for DDPG and TD3, and the *temperature coefficient* along with the *learning rate* for SAC. This was done to avoid using Equation 5.7, due to the computational complexity associated with our high-dimensional continuous problem setting. Note that the codebase for our work in this chapter is available at: `https://github.com/nkhumise-rea/mutli-dimensional_performance_metric_for_RL.git`.

6.3.3 Learning Curves and Performance Analysis

The results presented in Figure 6.2 are for the algorithms with tuned hyperparameter settings for the task. The optimal hyperparameter settings were identified through grid search sweeping (discussed later in Section 6.3.5) and they are specified in Table 6.1. Note that only hyperparameters which were involved in the grid search are enlisted, while the remaining — which are unchanged and shared by all the algorithms — were presented in Section 6.3.1.

Six training runs with different random seeds were carried out per algorithm to produce the learning curves in Figure 6.2. The learning curves are a typical way of presenting algorithm performance in RL, and hence we utilise them as baseline to compare with our multi-faceted performance metric. From Figure 6.2, we observe that SAC converges faster but at a slightly lower mean return than TD3. DDPG converges at the lowest mean return. In TD3, we notice a sudden drop followed by a recovery before the

training ends. This denotes a deviation from convergence in one of the runs, however the algorithm managed to find it again. Overall at the end of training, TD3 reaches the highest mean return, while DDPG possesses the highest return variance.

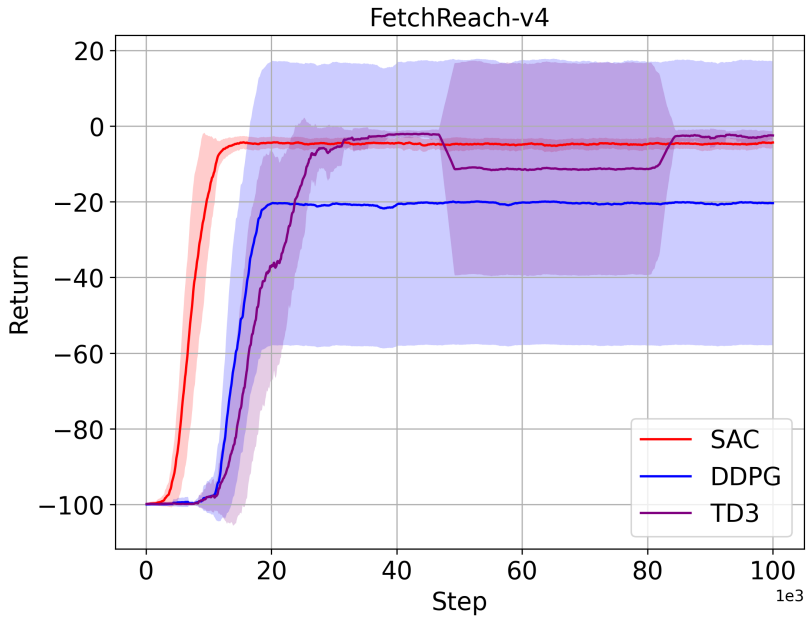


Figure 6.2: *Performance curves during learning of SAC, DDPG and TD3 algorithms for Reach task. 6 training trials were used to create the plots. The line in high contrast represents the mean, while the shaded region around the line is the standard deviation. The plot has been smoothed using a rolling mean with a window size of* 50.

Table 6.1: *Selected hyperparameter settings for algorithms SAC, DDPG and TD3. The remaining were not involved in grid search and are specified in Section 6.3.1. Note that the exploration noise σ was not involved in grid search as well, however it is enlisted in the table for convenience.*

Parameter	Value
SAC	
learning rate	0.001
temperature coefficient (α)	0.2
DDPG & TD3	
actor learning rate	0.001
critic learning rate	0.001
exploration noise (σ)	0.2

As a practitioner, the takeaway from Figure 6.2 is that TD3 is the best algorithm for the task once it has completed training because during deployment it is expected to output the highest mean return. However, care is warranted when limiting TD3 to relatively small number of training steps, since they may not be sufficient to allow the

algorithm to recover from losing its best convergent policy. If the learning speed and consistency is a priority, SAC would be the best algorithm to work with. Based on both mean performance and variance, DDPG would not be considered suitable for this task as it seems to be the least reliable.

Note that we receive no information about the quality of robot motion based on observing Figure 6.2 alone. On the contrary, during demonstrations, SAC consistently produced smooth robot motions, i.e. motions of the best quality; while TD3 and DDPG often produced motion with chatter (where the robot slightly quivers). This highlights a shortfall in using learning curves or cumulative rewards alone. A practitioner in robotics would prefer an algorithm with smoother robot motion, thus SAC would be selected over TD3 and DDPG. This is because the quality of motion in robotics is one of the highest priorities (Constantinescu & Croft, 2000; Paryanto et al., 2015; Guillén et al., 2020; Yuan et al., 2023).

6.3.4 Reliability Analysis

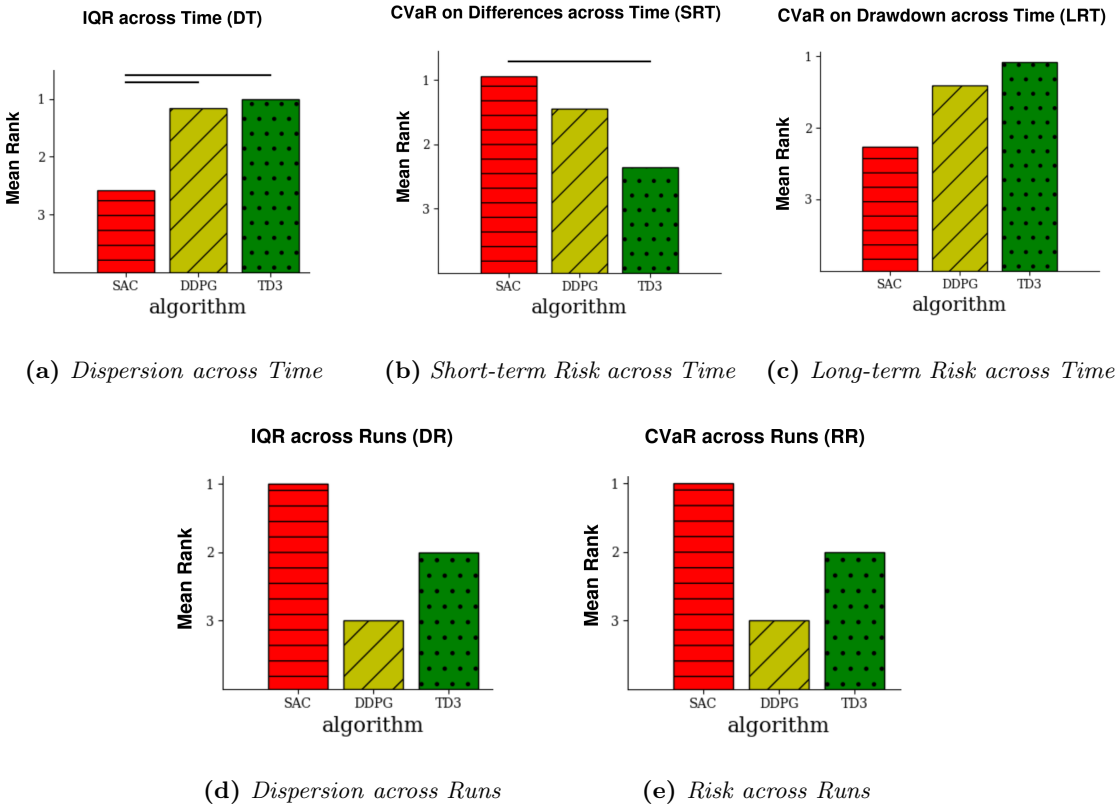
The results of the reliability analysis of the algorithms are displayed in Figures 6.3 and summarised in Table 6.2. IQR signifies the performance variability (dispersion), while CVaR denotes the expected performance value of the worst-performing run (or the worst performance drop within a run) with 5% probability of occurrence. In Figure 6.3 (a), we observe that TD3 has the lowest degree of variability in performance across time (within runs). It is followed closely by DDPG, while SAC has the highest performance variability. This means TD3 is locally consistent compared to SAC. On the contrary, SAC has the lowest risk of performance drop in the short-term (refer to Figure 6.3 (b)), while TD3 has the highest risk and DDPG is in the middle.

This helps contextualise the performance variability across time. It means the frequent performance changes in SAC are not necessarily random oscillations around a fixed value, but rather constitute a consistent upward trend. This denotes incremental improvement rather than noise. Compared to TD3, the infrequent performance changes consists of more frequent dips and hence carry a higher risk.

According to Figure 6.3 (c), in a long-term TD3 has the lowest risk of performance drop, followed by DDPG and SAC has the highest risk. This is not supported by the sudden drop in performance exhibited by TD3 in Figure 6.2. The sudden drop was undetected due to the reliability metrics ranking algorithms using median values (Chan et al., 2020). In our training runs, only a single trial exhibited this drop, and the median — being robust to outliers — essentially subdued the effect of that single unique trial. In this instance, the use of a median is disadvantageous since a valuable sample is deliberately ignored. This highlights a drawback in these metrics.

For variability across runs (Figure 6.3 (d)), SAC has the best performance consistency, followed by TD3, whereas DDPG has the worst. Moreover, SAC has the lowest risk of worst performance across runs (refer to Figure 6.3 (e)), while DDPG has the highest risk of worst performance. These results are consistent with, and supported by, the

performance bounds depicted in Figure 6.2. The results signify that across runs, SAC is the most reliable algorithm for the task, followed by TD3, and DDPG is the least reliable algorithm. Note that although the results are not statistically significant, they remain informative, particularly since they are consistent with patterns observed in Figure 6.2.



(a) *Dispersion across Time* **(b)** *Short-term Risk across Time* **(c)** *Long-term Risk across Time*

(d) *Dispersion across Runs* **(e)** *Risk across Runs*

Figure 6.3: *Reliability metrics rankings for the FetchReach-v4 task during training. Note that rank 1 always indicates better metric outcome. (a) - (c) rankings of metrics measured across time within runs, while (d) and (e) are rankings of metrics evaluated across runs. Note that statistically significant differences in ranking have a black horizontal line above the bars, its absence indicates non-significance. Note that the corresponding raw results are presented in Appendix F*

The key implication of these reliability results can be elucidated better using Table 6.2. The table portrays the ranking order of performance reliability per metric. Across all the reliability metrics, SAC is most reliable relative to its counterparts, followed by TD3, while DDPG is the least reliable. Unlike with learning curves, which only provide a picture across runs, the reliability metrics also provide insights about reliability across time within runs. For example, SAC is revealed to vary frequently with a positive trend in comparison to TD3 and DDPG. Furthermore, this implies that if training (within a run) is halted arbitrarily, then the observed performance of the algorithm is most likely not a representative of the final performance once training is complete. This

information is useful to practitioners because (1) conclusions cannot be made about the algorithm early or mid-training but until converge is reached; and (2) tuning the algorithm based on early returns can introduce bias.

Table 6.2: *Summary of reliability ranking results across the metrics. Note that DT = IQR across time, DR = IQR across runs, RR = CVaR across runs, SRT = CVaR on Differences across time, and LRT = CVaR on Drawdown across time. Moreover, the star ($\star$) denotes the algorithm outranks its counterpart with statistical significance.*

Rank	DT	SRT	LRT	DR	RR	Overall
1	TD3*	SAC*	TD3	SAC	SAC	SAC
2	DDPG	DDPG	DDPG	TD3	TD3	TD3
3	SAC	TD3	SAC	TD3	DDPG	DDPG

6.3.5 Hyperparameter Sensitivity Analysis

To explore the hyperparameter space, we performed grid search sweeps across selected hyperparameters using values specified in Table 6.3. During the hyperparameter sensitivity analysis, we varied two most influential hyperparameters per algorithm over three possible values to reduce computational costs. In SAC, the learning rate (of both the actor and critic networks) and the temperature coefficient α impact the performance of the algorithm the most, so they were varied. While for TD3 and DDPG, the actor and critic learning rates were varied since they have the most impact on performance. Often the exploration noise for these algorithms is kept constant (Lillicrap et al., 2016; Fujimoto et al., 2018; Yuan et al., 2022) and so we chose it be $\sigma = 0.2$, similar to Lillicrap et al. (2016).

Table 6.3: *Hyperparameter search space for DDPG, TD3 and SAC algorithms.*

Algorithm	Hyperparameter	Values
DDPG	actor learning rate	$\{10^{-4}, 10^{-3}, 10^{-2}\}$
	critic learning rate	$\{10^{-4}, 10^{-3}, 10^{-2}\}$
TD3	actor learning rate	$\{10^{-4}, 10^{-3}, 10^{-2}\}$
	critic learning rate	$\{10^{-4}, 10^{-3}, 10^{-2}\}$
SAC	learning rate	$\{10^{-4}, 10^{-3}, 10^{-2}\}$
	α coefficient	$\{0.1, 0.2, 0.5\}$

Each hyperparameter setting was evaluated over 50 runs to determine performance scores per environment. In our experiment, five environments corresponding to horizon lengths $\{25, 50, 75, 100, 125\}$ were employed to determine the hyperparameter sensitivity presented in Table 6.4. The results reveal that TD3 is relatively the most sensitive algorithm by substantial margins, while SAC is the least sensitive. DDPG is expectedly more sensitive than SAC, thus being consistent with literature ()Haarnoja18, but

by moderate margins. It has to be noted that the result outline the robustness of the algorithm with respect to horizon lengths of the task. Further, the results in Table 6.4 suggest that TD3 has the poorest horizon generalisation capabilities in comparison to its counterparts.

Note that the learning curves do not reveal anything about the robustness or brittleness of the algorithms with respect to their hyperparameters. The hyperparameter sensitivity metric Φ provides RL practitioners insights about the tuning effort required to determine the most suitable hyperparameter settings per environment. In our case, TD3 is relatively more brittle and would require the most tuning effort to attain peak scores, while SAC would maintain relatively the same peak scores with minimal or no tuning of hyperparameters.

Table 6.4: *Hyperparameter sensitivity of algorithms for the reach task across varying horizon lengths.*

Algorithm	Φ $(\times 10^{-3})$
TD3	30.03±11.11
DDPG	2.650±0.846
SAC	1.999±0.466

6.3.6 Exploration Analysis

To carry out experiments in this section, we utilised and trained cVAEs with hyperparameters listed in Table 6.5. Each policy π had a dedicated cVAE that was trained with the corresponding policy dataset $\mathcal{D}_\pi = \{(s_{(i)}, a_{(i)})\}_{i=1}^{m}$ of size $m = 2 \times 10^4$. These many samples were necessary to create high data density that can mitigate the effects of the curse of dimensionality during the probability density estimation (Vandermeulen et al., 2024). This is because the state-action space dimension of our environment is 14.

The cVAEs are the conditional distributions $\beta_a(S) = P(S \mid A = a)$, from which 500 state samples were collected for each action and used to compute the inner Wasserstein distance in Equation 6.1. This inner Wasserstein distance represents the distance between actions. To compute the outer Wasserstein distance, Algorithm 3 was employed with sample size $N = 128$ and number of batches $B = 3$. Note that details about the training of cVAE models and the assessment of samples they generated are presented in Appendix F.

Figure 6.4 denotes the exploratory behaviour of the algorithms in the policy space during training, hence we often refer to it as *policy evolution plot(s)*. The 2D versions of the policy evolution plots are displayed in Figure 6.5. The policy evolution plots show that the policy trajectory for SAC starts with larger jumps between policies and tapers as the optimal policy is approached. We also observe a noticeable reduction in distance-to-optimal followed by minimal fluctuations in Figure 6.5 (b) (2^{nd} column). This makes the trajectory smoother and less meandering than the algorithm's counterparts.

Table 6.5: *cVAE Hyperparameters.*

Parameter	Value
learning rate	0.001
number of training epochs	25
number of hidden layers (all networks)	2
number of hidden units per layer	32
number of samples per minibatch	512
latent variable dimension	6
nonlinearity	ReLU

Moreover, the OMR plot in Figure 6.5 (b) (3^{rd} column), shows that SAC made less wasteful and more direct policy transitions, than DDPG and TD3, as it approached the optimal policy. All these results indicate that SAC explored the policy space more efficiently that DDPG and TD3. The policy trajectories for DDPG and TD3 look somehow similar. This is not surprising since TD3 is a derivative of DDPG (as discussed in Chapter 2). Nevertheless, the magnitude of jumps between policies is maintained without significant reduction. This shows that the policy trajectory maintains the same level of meandering, regardless of how far the agent is from the optimal policy.

In this task environment, the true optimal policy is challenging to identify and is not a known priori. So, we approximate optimality empirically by choosing the converged policy of each algorithm (i.e. after training), provided it has stable performance and satisfies the task success criteria. The converged policy is treated as an estimate of the optimal policy for evaluation purposes.

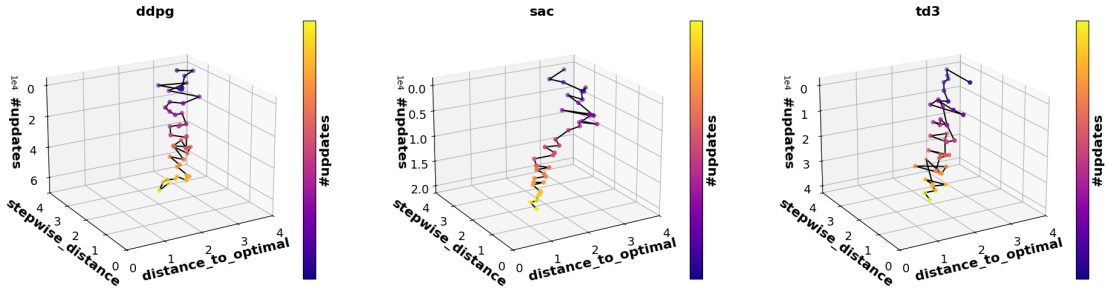


Figure 6.4: *3D plots of distance-to-optimal and stepwise-distance vs. number of updates for DDPG, SAC and TD3 algorithms. See Appendix F for further details about the process behind devising these results.*

Recall from Figure 6.3 (b) that SAC had the lowest risk of performance drops within runs. Now, looking at Figure 6.4, we notice that the algorithm makes small jumps between policies almost midway through training. These figures support the idea that the small jumps are supposedly made to prevent short-term performance drops by ensuring that the next policy has slightly better performance than the previous one. This exploration strategy maximises long-term returns while mitigating the likelihood

of performance degradation. It explains why SAC learns more stably, where it will not update from a near-optimal policy to an extremely worse policy — unlike TD3, as we observed in Figure 6.2.

The aforementioned insight elucidates how the exploration strategy of SAC avoids catastrophic performance drops, and it would not have been understood solely from the learning curves. This underscores the benefit of a multi-faceted evaluation of algorithmic performance. From Table 6.6, we expectedly observe that SAC has the lowest ESL value and highest OMR value, which denotes superior exploration efficiency in the task. By examining OMR in both Table 6.6 and Figure 6.5 (3^{rd} column), we notice that TD3 and DDPG exhibit a drop in OMR just before reaching the optimal policy. This highlights that even in the proximity of the optimal policy, the algorithms still meandered. This persistent meandering explains why these algorithms are prone to catastrophic performance drops or high performance variability (as depicted in Figure 6.2).

Unfortunately, we cannot glean much information about the exploration strategy of algorithms from the learning curves alone (as previously discussed in Chapter 3). However, by observing multiple dimensions of performance, as done here, we begin to uncover some underlying mechanisms behind the algorithms. An RL practitioner or researcher can benefit from this added layer of interpretability provided by a multi-faceted performance metric. In the next section, we consolidate the results across all the dimensions of performance to make a comprehensive comparison analysis.

Table 6.6: *Evaluation of RL algorithms in the FetchReach-v4 environment (over 5 runs). Note that values in* ***bold*** *signify best metric performance.*

Algo.	ESL	OMR
DDPG	34.15±2.93	0.514±0.05
SAC	**27.06±3.21**	**0.559±0.07**
TD3	30.73±0.96	0.528±0.07

6.3.7 Consolidation and Comparison Analysis

After conducting analyses across various evaluation metrics, the performance results of the algorithms can be summarised as depicted in Table 6.7. Since the evaluation metrics have incommensurable units, we resort to ranking to assist with interpretation of performance per dimension. As suggested by Whiteson et al. (2011), we use *sign test* (Salzberg, 1997) for each pair of algorithms to determine which algorithm has best overall performance. In the sign test, a pairwise comparison between algorithms (e.g. A and B) over each dimensional metric is performed. $'+'$ sign is issued if A > B, $'-'$ sign if A < B and 0 if a tie. The number of + and − signs are counted and a binomial distribution is employed to observed if the number of + signs differs significantly. In short, the sign test evaluates the hypothesis that $\mathbb{P}(A > B) > 0.5$ (Whiteson et al., 2011). The results of the sign test are presented in Table 6.8.

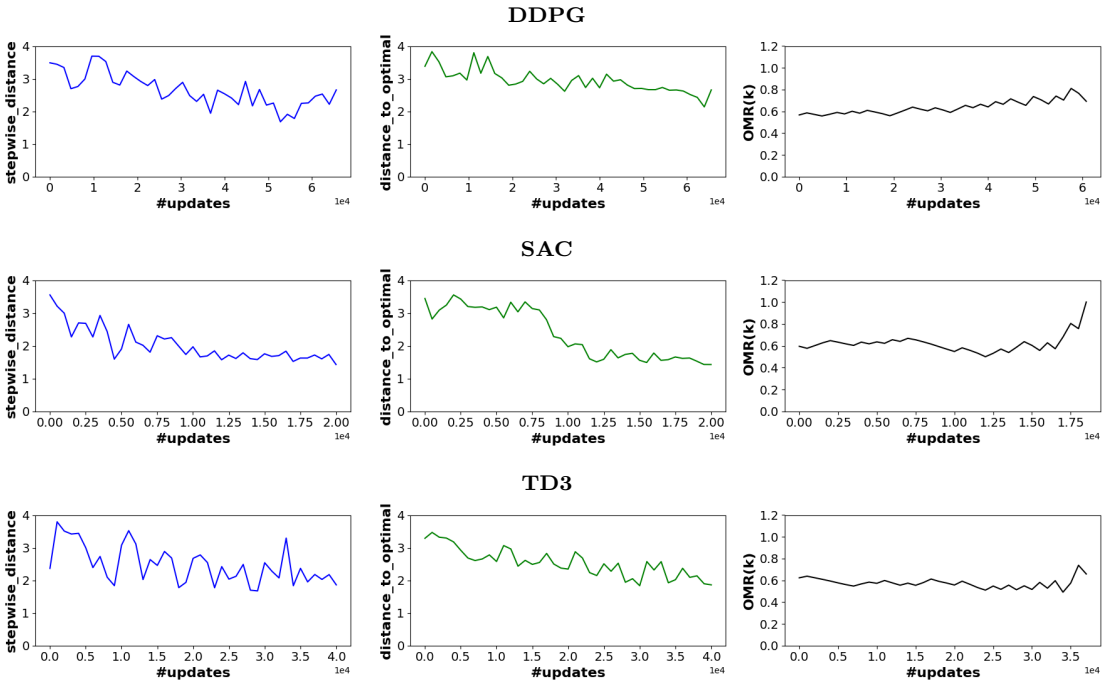


Figure 6.5: *Plots in the first column are stepwise-distance vs. number of updates, second column distance-to-optimal vs. number of updates, and third column OMR(k) vs. number of updates. The plots in the row belong to algorithms in the following order from top to bottom: DDPG, SAC and TD3.*

Table 6.7: *Comparison of algorithms DDPG, TD3 and SAC in the FetchReach-v4 environment across multi performance dimensions.*

Performance Dimension	Metric	Ranking
Long-term consequences	Mean Return	TD3 > SAC > DDPG
	Motion quality	SAC > TD3 > DDPG
Robustness	Reliability	SAC > TD3 > DDPG
	Hyperparameter sensitivity	SAC > DDPG > TD3
Exploration	ESL	SAC > TD3 > DDPG
	OMR	SAC > TD3 > DDPG

Table 6.8: *Results of the sign test along with statistical testing for comparative study of algorithms SAC, DDPG and TD3 across six performance metrics. Statistical significance is with* $p < 0.01$.

	SAC vs TD3	SAC vs DDPG	TD3 vs DDPG
Probability	0.833	1.0	0.833
p-value	0.110	0.015	0.110

From Table 6.8, we note that under the six performance metrics, SAC outperforms both TD3 and DDPG, while TD3 outperforms DDPG for the selected task (i.e. FetchReach-v4). It is almost indisputable that SAC is better than DDPG in this setting, since it outperforms DDPG across all performance dimensions. Note that the evidence is insufficient to conclude statistical significance due to the small number of metrics. However, this does not make the results less practically significant (Agarwal et al., 2021).

6.4 Discussion

The evaluation of a multi-dimensional performance across exploration, robustness and long-term consequences (behaviour) was conducted for algorithms SAC, TD3 and DDPG in a reaching task using a 7-DoF robotic arm. We made the following observations:

Long-term consequences (behaviour). The learning curves in conjunction with behavioural descriptions of the agent revealed that SAC generally learns faster than both TD3 and DDPG. However, TD3 can reach the highest return score compared to SAC and DDPG, even though it is prone to performance drops. DDPG generally learns faster than TD3 and has relatively the worst performance variance. With the assistance of the behavioural descriptions of the agent, we learned that SAC outputs smoother robot motions than its counterparts.

Robustness. The reliability analysis results unveiled that SAC exhibits higher performance variability across time (within runs), but it is not due to noise. Rather, it constitutes a consistent upward trend that indicates incremental performance improvement. SAC is relatively the most reliable algorithm across runs with better consistency and less risk of worst-performance run compared to TD3 and DDPG. Overall, TD3 comes second best behind SAC, and DDPG is relatively the least reliable algorithm.

Exploration. We learned from exploration analysis that the exploration strategy of SAC maximises long-term returns while minimising the risk of ruin by adjusting to small jumps between policies midway through training. This makes SAC learn more stably. In contrast, TD3 and DDPG maintain the same magnitude of jumps between policies and persist to meander even in proximity to the optimal policy. This makes the algorithms susceptible to catastrophic performance drops and high performance variability. Behind SAC, TD3 had a more efficient exploration strategy than DDPG.

Overall. Across exploration, robustness and long-term consequences, SAC comes out as relatively the most *efficient RL algorithm* compared to TD3 and DDPG. Not only did we witness SAC's superiority across the performance dimensions, but using this multi-faceted approach, we improved the interpretability of comparative analyses of RL algorithms. This is something we aimed to achieve, as indicated from the beginning in Chapter 1.

6.5 Concluding Remarks

In this chapter, we integrated various metrics across dimensions of algorithm performance into an evaluation framework and demonstrated its utility in a complex environment. Compared to using solely cumulative rewards (learning curves), we showed that there is practical value in employing a multi-dimensional performance metric in RL — as it has been suggested in the literature (Whiteson et al., 2011; Henderson et al., 2018; Osband et al., 2019; Adkins et al., 2024). This framework can enhance our understanding and interpretability of RL methods. With this notion established, we discuss the entire thesis work in the next Chapter.

Chapter 7

Discussion

7.1 Introduction

Reinforcement learning (RL) has become a crucial framework for developing autonomous agents (Kulkarni & Patil, 2025). When integrated with deep neural networks, collectively called *deep RL*, it has led to significant advances across numerous areas of application, including robotics (Zhang & Mo, 2021). Major challenges in deep RL stem from its lack of a complete theory (Osband et al., 2019; Laidlaw et al., 2023) and poor reproducibility (Henderson et al., 2018). These subvert the interpretability of RL methods and, consequently, constrain confidence in their practical deployment (Grothoff et al., 2022).

In addressing this, researchers have called for the investigation of new evaluation metrics (Whiteson et al., 2011; Henderson et al., 2018; Dulac-Arnold et al., 2019; Adkins et al., 2024) that can improve the understanding (Conserva & Rauber, 2022) and interpretability of comparative analyses of RL algorithms. The reason is that the current use of a single-faceted performance metric, such as cumulative reward, across several benchmarks offers limited insight about the underlying behaviour of RL methods (Adkins et al., 2024). This motivated us to investigate methods that can capture the multi-faceted performance of RL algorithms.

Naturally two questions arise: *What can be described as reasonable dimensions of agent performance?* and *What is an appropriate empirical methodology to evaluate the algorithms?* Since an efficient RL agent must handle exploration, generalisation and long-term consequences (Osband et al., 2019), we appoint these three factors as the main dimensions of agent performance. However, measuring performance across these dimensions presents the following challenges:

1. *Generalisation* in RL is difficult to assess directly (Kirk et al., 2023), however analysing *Robustness* and *Task Complexity* can be utilised as a basis for understanding it (Kuzborskij & Lampert, 2018).

2. Consensus on quantitatively evaluating *Exploration* across various algorithms remains absent (Seijen et al., 2020; Amin et al., 2021; Ladosz et al., 2022).

Addressing these challenges is part of investigating an appropriate empirical framework to evaluate RL algorithms. Moreover, we replace *generalisation* with *robustness*, to simplify the evaluation framework. With this established, the central **research question** is:

> Can a rigorous evaluation methodology that captures *exploration, robustness*, and *long-term consequences (behaviour)* of RL algorithms enhance the understanding and interpretability of their comparative analyses?

The thesis addressed this question by following a research methodology outlined in the next section (Section 7.2). In Section 7.3, we summarise the contributions that were consequential of this research, and discuss recommendations for future work in Section 7.4 prior to making concluding remarks in Section 7.5.

7.2 Summary of the Thesis work

We began our study by investigating suitable metrics for assessing *exploration*, *robustness* and *long-term consequences* of RL algorithms. This was mainly carried out across the thesis chapters 3 - 6 as follows:

Chapters 3: Studying exploration in RL. We developed theoretically grounded metrics for evaluating exploration, namely *Effort of Sequential Learning* (ESL) and *Optimal Movement Ratio* (OMR). ESL quantifies the relative effort (or cost) of the exploratory process of the algorithm to learn the environment. It is the relative distance an algorithm travels compared to the shortest path from the initial to the optimal policy during training. OMR measures the fraction of policy model updates that contribute towards closing the gap between the optimal policy and the latest agent policy, during training.

We showed that the space of occupancy measures (probability space) is a differentiable manifold and has a one-to-one relationship with the policy space. This enabled us to measure distances (using the Wasserstein metric) among policies in the policy space using their corresponding occupancy measures. Furthermore, it enabled visualisation of the policy evolution during training — which revealed how algorithms explore the policy space.

Chapters 4: Task complexity. In the Introduction chapter (i.e. Chapter 1), we learned that generalisation is challenging to examine directly. So, we assessed its prerequisites, viz. *task complexity* and *robustness*, to gain a better understanding of it. We highlighted that task complexity is necessary for informing generalisation in RL, since it is key for task selection in both curriculum learning and benchmarks. However, it is non-trivial to develop a metric for task complexity and current approaches can

often fail to be reliable.

We demonstrated using a *1-link* and *2-link* manipulators, with known relative complexity (based on control theory), that metrics *Policy Information Capacity* (PIC) and *Policy-Optimal Information capacity* (POIC) can yield results that contradict the known relative complexity. Furthermore, we developed a simple framework for assessing task complexity metrics, prior to being applied to sophisticated benchmarks. Our work showed that reliably evaluating task complexity in RL remains nuanced and unresolved.

Chapters 5: Robustness analysis. We analysed robustness of algorithms from the perspective of reliability and hyperparameter sensitivity. For capturing reliability of performance, we utilised interquartile range (IQR) and conditional value at risk (CVaR) to quantify performance variability and risk of performance drop across time or runs during training. For assessing hyperparameter sensitivity, we employed a hyperparameter sensitivity metric developed by Adkins et al. (2024). The metric is defined as the difference between an algorithm's per-environment tuned score and its cross-environment tuned score. Moreover, we assessed the relative contribution of hyperparameters in the algorithm's performance variance using variance-based analysis.

In addition to robustness, we evaluated the performance of SAC and DDPG algorithms across a set of reaching tasks with various robotic manipulator configurations. We demonstrated that in capturing *long-term consequences (behaviour)* of an agent, it is crucial to report achieved cumulative rewards in conjunction with the behaviour of the agent. This contextualises the cumulative rewards. The reason is that cumulative rewards do not reliably represent the desired behaviour of the agent (Henderson et al., 2018; Yau, 2024), especially for environment where rewards are not an innate property (Andrychowicz et al., 2017; Ecoffet et al., 2019).

Chapters 6: Multi-dimensional performance metric. After successfully identifying metrics for *exploration*, *robustness* and *long-term consequences*, we deployed them in a high dimensional continuous problem setting (controlling a 7-DoF robotic arm). The setting posed computational challenges mainly for adapting the exploration and hyperparameter sensitivity metrics, owing to its high dimensional and continuous state-action space. To alleviate this, we applied cVAEs to estimate action-dependent state distributions — which allowed us to compute the path distances traversed by the algorithms during training as part of assessing exploration. Furthermore, we used statistical approximation methods to calculate the Wasserstein distances that represent these path distances.

For hyperparameter sensitivity, we constructed proxy environments by varying the planning horizon of a chosen task to attain a computationally cheaper environment set. This was essential because examining hyperparameter sensitivity across multiple environments is computationally costly when using distinct tasks. Of course, our methods to adapt the exploration and hyperparameter sensitivity metrics (in this high-dimensional continuous setting) encompass some loss of accuracy and rely on additional

underlying assumptions. Nevertheless, we managed to showcase the utility of a multi-faceted evaluation framework for RL algorithms.

Addressing the Research Question. The answer to the **research question** is: *Yes, our multi-dimensional performance evaluation framework improves the understanding and interpretability of comparative analysis of RL algorithms.* We demonstrated this in Chapter 6 by employing SAC, DDPG and TD3 algorithms. We derived interesting insights such as how SAC's exploration strategy maximises long-term returns while minimising the risk of ruin, by making small conservative jumps between policies midway through training. This enhances the learning stability of the algorithm. Additionally, we learned that SAC is relatively the most robust algorithm with respect to its hyperparameters, while TD3 is the least robust algorithm. Overall for the *FetchReach-v4* reach task we utilised, *SAC stands out as a more efficient algorithm than TD3 and DDPG.*

7.3 Summary of contributions

In this thesis work, we made the following contributions in the field of RL:

1. We introduced a framework for comprehensively analysing RL algorithms across dimensions of performance, namely: *exploration, robustness* and *long-term consequences*. This multi-faceted framework offers an approach to enhancing current insights into RL algorithms.

2. We demonstrated that current quantitative metrics of task complexity in deep RL fail to capture the correct hardness of some simple robotic tasks.

3. We developed metrics for evaluating exploration. We provided theoretical justification and validated them empirically. Furthermore, we extended the metrics to high-dimensional continuous settings.

4. We conducted a detailed comparative analysis of SoTA RL algorithms — namely:- SAC, DDPG and TD3 — in a robotic task using the proposed multi-faceted evaluation framework. From this, we learned that SAC is relatively the most efficient algorithm for the task.

7.4 Directions for Future work

In this section, we highlight directions of future work in two ways. One is about how the work presented in this thesis can be improved, while the other is about possible research avenues in which extensions of this thesis work can be valuable. It should be noted that this section is both speculative and reflective.

7.4.1 Areas of Improvement

Multi-dimensional performance framework. The multi-dimensional performance framework we proposed can benefit from more metrics beyond exploration, robustness and long-term consequences. For example, metrics that can evaluate sample efficiency, various types of generalisation, safety, computational efficiency, etc., can improve the framework. The drawback is that adding more metrics increases the computational complexity of the framework. This creates a trade-off between the level of analytical detail and the computational cost required to obtain it. Additionally, it underscores the need for metrics that can be assessed easily with little computational overhead.

Hyperparameter sensitivity. Examining the hyperparameter sensitivity of algorithms especially in high-dimensional continuous domains has high computational requirements (as discussed in Section 6.2.2). We managed to lower computational costs by considering a set of tasks instantiated from the same environment but with distinct planning horizons (i.e. homogeneous tasks). While this approach worked, it does not guarantee that the findings will generalise to heterogeneous tasks (i.e. tasks with distinct environments). We encourage using both homogeneous and heterogenous tasks for hyperparameter sensitivity analysis. In that manner, the analysis will be more exhaustive.

Exploration. Adapting ESL and OMR metrics to high-dimensional continuous domains requires estimating a complex conditional probability distribution of states given actions (as discussed in Section 6.2.1). We employed conditional variational autoencoders (cVAEs), however alternative generative methods such as diffusion models (Liu et al., 2024; Maggiora et al., 2023) or variational flows (Xu & Campbell, 2023) could also be considered. These methods have been shown to attain higher-fidelity density estimation than cVAEs in some high-dimensional or multimodal settings as reported in (Bhattacharyya et al., 2019; Kingma et al., 2021).

7.4.2 Prospective Avenues

Bisimulation and Occupancy couplings in exploration. In hierarchical reinforcement learning (HRL), action abstractions can improve exploration of the environment (Nachum et al., 2019). Similarly, state abstractions — using e.g. bisimulation to reduce the state space size by aggregating bisimilar states (Ferns et al., 2004) — simplify exploration for the agent. The reason being that the agent explores at a higher abstract level in a smaller state space. These approaches underscore the impact of state and action abstraction on exploration. However, challenges exist in selecting abstractions which are sufficiently expressive to model complex environments (Hutsebaut-Buysse et al., 2022). Additionally, state and action abstractions are typically used independently, as in bisimulation (Ferns et al., 2004) versus options (Sutton et al., 1999b) frameworks, and to our knowledge, a convincing joint state-action abstraction has not yet appeared in the HRL literature.

Two states are bisimilar when they have identical reward and transition probabili-

ties (Givan et al., 2003). A bisimulation metric measures how closely bisimilar any two states are, allowing states with minimal bisimulation distance to be aggregated into a single abstract state. Calo et al. (2024) showed that the bisimulation distance between two states is given by the inner product of a cost matrix (reflecting reward differences between state pairs) and an *occupancy coupling*, which is the probability distributions over state-state pairs.

While occupancy measures $v_\pi(s, a)$ (discussed in Chapter 3) capture state-action pair visits, *occupancy couplings* capture state-state pair visits from two markov chains that are being compared Calo et al. (2024). Hence, we propose developing joint state-action abstractions by extending bisimulation from purely *state-based occupancy couplings* to *state-action based occupancy couplings*, on which we can apply our framework for computing the path lengths taken by RL algorithms during training — i.e. OTDD in the RL domain, discussed in Section 3.5.1. It will be interesting to investigate analytically whether a joint state-action abstraction can be achieved by employing a bisimulation-like approach using our framework to analyse the space of state-action occupancy measures. This can contribute to enhancing exploration strategies of RL agents.

Online Adaptation of exploration. Adapting the exploration strategy *online* based on new observations, akin to human beings, is an interesting idea that is actively being explored including in works such as (Wagenmaker et al., 2025). It aims to avoid local optima in the policy space by making informed transitions between policies. This can potentially improve exploration strategies and make them more efficient than the *uninformed* novelty-seeking exploration strategies (Wagenmaker et al., 2025). The ESL metric can be employed to inform the policy updates and encourage/discourage the direction of the policy updates.

This is similar to how the KL-divergence ($\mathrm{KL}(\pi_{\theta_{old}} \parallel \pi_{\theta_{new}})$) and policy ratio ($\frac{\pi_{\theta_{old}}}{\pi_{\theta_{new}}}$) are used in *Trust Region Policy Optimisation* (TRPO) (Schulman et al., 2015) and *Proximal Policy Optimisation* (PPO) (Schulman et al., 2017b) algorithms, respectively, to guide policy updates. For example, we can define a loss function for the policy model that minimises ESL values. This would discourage meandering in the policy space and promote policy transitions that lead directly to the optimal policy.

Generalisation. Generalisation in RL is a broad concept that has numerous facets including but not limited to generalising over *horizons*, *states of an MDP*, *across structurally similar MDPs*, and *across structurally distinct MDPs*. Currently, there is no comprehensive work that describes these facets in detail, however the work that comes close to doing this is (Kirk et al., 2023). Kirk et al. (2023) contextualise generalisation across structurally similar and distinct MDPs, while excluding generalisation over horizons and states of an MDP. Under generalisation, assessing task complexity is still an open problem, and we know that curriculum learning (CL) and benchmarks in RL can remain limited in scope without reliable metrics for task complexity (Duan et al., 2016; Narvekar, 2021). Consequently, this inhibits the study on generalisation espe-

cially in CL — which is a promising avenue for advancing generalisation capabilities of RL algorithms.

Long-term consequences. Currently, the best metric for evaluating long-term behaviour of RL agents is expected returns (as discussed in Chapter 1). As highlighted by Amodei et al. (2016), the credibility of the expected return remains underexplored especially in domains where rewards are not an innate property. We have seen that in such domains, e.g. robotics, expected returns do not faithfully represent the quality of agent behaviour (Andrychowicz et al., 2017). While we reported agent behaviour along with expected returns for context (following the recommendation by Henderson et al. (2018)), we feel more can be done.

Specifically, there is a need to quantify uncertainties introduced by reward functions. Error bounds between a selected reward function and the desired agent behaviour can be of value to derive. Typically, researchers and practitioners try multiple reward functions, with careful hand-crafting or engineering. However, there is no guarantee on how reliable these would be. Furthermore, communicating these uncertainties can help practitioners in selecting and comprehending the benefits and drawbacks of reward functions, without necessarily seeing agent demonstrations.

7.5 Concluding remarks

This chapter reflected on the entire thesis work and elucidated how the work addressed the central and associated research questions. A multi-dimensional performance metric that evaluates the agent across exploration, robustness and long-term behaviour was successfully developed, and indeed it can enhance the understanding and interpretability of comparative studies in RL algorithms. Definitely, there is room for improvement (as highlighted above). We hope that the work in this dissertation can serve as a basis for further development and deployment of multi-dimensional performance metrics in RL.

Bibliography

M. A. Abdullah, H. Ren, H. B. Ammar, V. Milenkovic, R. Luo, M. Zhang, and J. Wang. Wasserstein robust reinforcement learning. *arXiv preprint arXiv:1907.13196*, 2019.

D. Abel, C. Allen, D. Arumugam, D. E. Hershkowitz, M. L. Littman, and L. L. S. Wong. Bad-policy density: A measure of reinforcement learning hardness. *arXiv preprint arXiv:2110.03424*, 2021.

D. Abel, M. K. Ho, and A. Harutyunyan. Three dogmas of reinforcement learning. In *Reinforcement Learning Conference (RLC)*, 2024. URL `https://arxiv.org/pdf/2407.10583`.

C. Acerbi and D. Tasche. Expected shortfall: a natural coherent alternative to value at risk. *Economic notes*, 31(2):379–388, 2002.

J. Adkins, M. Bowling, and A. White. A method for evaluating hyperparameter sensitivity in reinforcement learning. *Advances in Neural Information Processing Systems*, 37:124820–124842, 2024.

A. Agarwal, N. Jiang, S. M. Kakade, and W. Sun. Reinforcement learning: Theory and algorithms, 2022. URL `https://rltheorybook.github.io/`.

R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

O. Ahmed, F. Träuble, A. Goyal, A. Neitz, Y. Bengio, B. Schölkopf, M. Wüthrich, and S. Bauer. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. *arXiv preprint arXiv:2010.04296*, 2020.

M. Aleksandrowicz and J. Jaworek-Korjakowska. Metrics for assessing generalization of deep reinforcement learning in parameterized environments. *JAISCR*, 14(1):45–61, 2023.

A. Alemi, B. Poole, I. Fischer, J. Dillon, R. A. Saurous, and K. Murphy. Fixing a broken elbo. In *International conference on machine learning*, pp. 159–168. PMLR, 2018.

E. Altman. *Constrained Markov Decision Processes*. Routledge, 1999.

D. Alvarez-Melis and N. Fusi. Geometric dataset distances via optimal transport. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

S. Amin, M. Gomrokchi, H. Satija, H. van Hoof, and D. Precup. A survey of exploration methods in reinforcement learning. *arXiv preprint arXiv:2109.00157*, 2021.

D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, OpenAI A. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

B. D. Argall. Autonomy in rehabilitation robotics: An intersection. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):441–463, 2018.

S. Arimoto. Stability and robustness of pd feedback control with gravity compensation for robot manipulator. In *Robotics research, First international symposium, 1986*, pp. 783–799, 1986.

K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017. URL `https://arxiv.org/pdf/1708.05866`. IEEE Signal Processing Magazine, Special Issue on Deep Learning for Image Understanding.

M. G. Azar, I. Osband, and R. Munos. Minimax regret bounds for reinforcement learning. In *International conference on machine learning*, pp. 263–272. PMLR, 2017.

D. Bank, N. Koenigstein, and R. Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pp. 353–374, 2023.

D. Basu, X. Wang, Y. Hong, H. Chen, and S. Bressan. Learn-as-you-go with megh: Efficient live migration of virtual machines. *IEEE Transactions on Parallel and Distributed Systems*, 30(8):1786–1801, 2019.

D. Basu, P. Senellart, and S. Bressan. Belman: An information-geometric approach to stochastic bandits. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part III*, pp. 167–183. Springer, 2020.

M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 1471–1479. Curran Associates, Inc., 2016.

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*, 47:253–279, 2013.

R. Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pp. 679–684, 1957.

C. C. Beltran-Hernandez, D. Petit, I. G. Ramirez-Alpizar, and K. Harada. Variable compliance control for robotic peg-in-hole assembly: A deep-reinforcement-learning approach. *Applied Sciences*, 10(19):6923, 2020a.

C. C. Beltran-Hernandez, D. Petit, I. G. Ramirez-Alpizar, T. Nishi, S. Kikuchi, T. Matsubara, and K. Harada. Learning force control for contact-rich manipulation tasks with rigid position-controlled robots. *IEEE Robotics and Automation Letters*, 5(4): 5709–5716, 2020b.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

D. J. Benjamin, J. O. Berger, M. Johannesson, B. A. Nosek, E. J. Wagenmakers, R. Berk, K. A. Bollen, B. Brembs, L. Brown, C. Camerer, et al. Redefine statistical significance. *Nature human behaviour*, 2(1):6–10, 2018.

D. P. Bertsekas and J. N. Tsitsiklis. Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE conference on decision and control*, volume 1, pp. 560–564. IEEE, 1995.

A. Bhattacharyya, M. Hanselmann, M. Fritz, B. Schiele, and C. N. Straehle. Conditional flow variational autoencoders for structured sequence prediction. *arXiv preprint arXiv:1908.09008*, 2019.

A. Billard and D. Kragic. Trends and challenges in robot manipulation. *Science*, 364, 2019.

L. Bingran, Z. Hui, Y. Peiqing, and W. Jinsong. Trajectory smoothing method using reinforcement learning for computer numerical control machine tools. *Robotics and Computer-Integrated Manufacturing*, 61:101847, 2020.

B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A. L. Boulesteix, D. Deng, and L. Marius. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2):e1484, 2023.

C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

H. Bojun. Steady state analysis of episodic reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 9335–9345. Curran Associates, Inc., 2020.

H. Bourel, O. Maillard, and M. S. Talebi. Tightening exploration in upper confidence reinforcement learning. In *International Conference on Machine Learning*, pp. 1056–1066. PMLR, 2020.

M. Bowling, J. D. Martin, D. Abel, and W. Dabney. Settling the reward hypothesis. In *International Conference on Machine Learning*, pp. 3003–3020. PMLR, 2023.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

J. Brink, B. Hinds, and A. Haney. Robotics repeatability and accuracy: another approach. *The Texas Journal of Science*, 56(2):149–157, 2004.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. URL `https://www.gymlibrary.dev/environments/classic_control/mountain_car/`.

Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1lJJnR5Ym`.

S. Calo, A. Jonsson, G. Neu, L. Schwartz, and J. Segovia-Aguas. Bisimulation metrics are optimal transport distances, and can be computed efficiently. *Advances in Neural Information Processing Systems*, 37:134345–134386, 2024.

G. Carlier, A. Galichon, and F. Santambrogio. From knothe's transport to brenier's map and a continuation method for optimal transport. *SIAM Journal on Mathematical Analysis*, 41(6):2554–2576, 2010.

S. C. Y. Chan, S. Fishman, J. Canny, A. Korattikara, and S. Guadarrama. Measuring the reliability of reinforcement learning algorithms. In *International Conference on Learning Representations*, 2020.

L. Chen, Z. Jiang, L. Cheng, A. C. Knoll, and M. Zhou. Deep reinforcement learning based trajectory planning under uncertain constraints. *Frontiers in Neurorobotics*, 16:883562, 2022.

S. Chen, B. Zhang, M. W. Mueller, A. Rai, and K. Sreenath. Learning torque control for quadrupedal locomotion. In *IEEE-RAS 22nd International Conference on Humanoid Robots (Humanoids)*, pp. 1–8, 2023a.

Y. H. Chen, W. T. Yang, B. H. Chen, and P. C. Lin. Manipulator trajectory optimization using reinforcement learning on a reduced-order dynamic model with deep neural network compensation. *Machines*, 11(3):350, 2023b.

S. Chiaverini. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Transactions on Robotics and Automation*, 13(3):398–410, 2002.

P. Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

F. H. Christopher and S. R. Patil. Identification and review of sensitivity analysis methods. *Risk analysis*, 22(3):553–578, 2002.

K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

M. Claesen and B. De Moor. Hyperparameter search in machine learning. In *Meta-heuristics International Conference (MIC)*, 2015.

K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. Quantifying generalization in reinforcement learning. In *International conference on machine learning*, 2019.

K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, 2020.

C. Colas, O. Sigaud, and P. Y. Oudeyer. A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms. *arXiv preprint arXiv:1904.06979*, 2019.

M. Conserva and P. Rauber. Hardness in markov decision processes: Theory and practice. *36th Conference on Neural Information Processing Systems*, 2022.

M. Conserva, R. Sasso, and P. Rauber. On the limits of tabular hardness metrics for deep rl: A study with the pharos benchmark. *arXiv preprint arXiv:2509.17092*, 2025.

D. Constantinescu and E. A. Croft. Smooth and time-optimal trajectory planning for industrial manipulators along specified paths. *Journal of robotic systems*, 17(5): 233–249, 2000.

C. Copot, C. Muresan, C.-M. Ionescu, S. Vanlanduit, and R. De Keyser. Calibration of ur10 robot controller through simple auto-tuning approach. *Robotics*, 7(3), 2018. URL `https://www.mdpi.com/2218-6581/7/3/35`.

P. Corke. *Robotics, vision and control*. Springer Berlin, Heidelberg, 2011.

E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016. URL `http://pybullet.org`. Software.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley Sons, Ltd, 2nd edition, 2006.

M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

K. Daaboul. Reinforcement learning: Dealing with sparse reward environments, 2020. URL `https://medium.com/@m.k.daaboul/dealing-with-sparse-reward-environments-38c0489c844d`.

M. Dalal, D. Pathak, and R. R. Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems*, 34:21847–21859, 2021.

R. de Lazcano, K. Andreas, J. J. Tai, S. R. Lee, and J. Terry. Gymnasium robotics. Software, 2024. URL `http://github.com/Farama-Foundation/Gymnasium-Robotics`. Software library of RL robotic environments.

M. P. Deisenroth, G. Neumann, and J. Peters. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

M. Delacre, D. Lakens, and C. Leys. Why psychologists should by default use welch's t-test instead of student's t-test. *International Review of Social Psychology*, 30(1): 92–101, 2017.

E. Derman and S. Mannor. Distributional robustness and regularization in reinforcement learning. *arXiv preprint arXiv:2003.02894*, 2020.

R. Devidze. *Reward design for reinforcement learning agents*. PhD thesis, Faculty of Mathematics and Computer Science, Saarland University, 2024. URL `https://arxiv.org/abs/2503.21949`. Doctoral thesis.

R. DeVore, B. Hanin, and G. Petrova. Neural network approximation. *Acta Numerica*, 30:327—-444, 2021.

D. Dewey. Reinforcement learning and the reward engineering principle. In *AAAI spring symposium*, 2014.

T. J. DiCiccio and B. Efron. Bootstrap confidence intervals. *Statistical science*, 11(3): 189–228, 1996.

T. Dick. Policy gradient reinforcement learning without regret. Master's thesis, Department of Computing Science, University of Alberta, 2015. MSc thesis.

C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

M. P. Drazin. Pseudo-inverses in associative rings and semigroups. *The American mathematical monthly*, 65(7):506–514, 1958.

N. du Preez-Wilkinson and M. Gallagher. Fitness landscape features and reward shaping in reinforcement learning policy spaces. In *International Conference on Parallel Problem Solving from Nature*, pp. 500–514. Springer, 2020.

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. *Proceedings of the 33 rd International Conference on Machine Learning*, 48:1329—1338, 2016.

G. Dulac-Arnold, D. Mankowitz, and T. Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881*, 2020.

A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

A. Edmondson and R. P. A. Petrick. Navigating errors: The tolerance of reinforcement learning algorithms to misleading heuristics. *Association for the Advancement of Artificial Intelligence*, 2025.

T. Eimer, M. Lindauer, and R. Raileanu. Hyperparameters in reinforcement learning and how to tune them. In *International conference on machine learning*, pp. 9104–9149. PMLR, 2023.

Í. Elguea-Aguinaco, A. Serrano-Muñoz, D. Chrysostomou, I. Inziarte-Hidalgo, S. Bøgh, and N. Arana-Arexolaleiba. A review on reinforcement learning for contact-rich robotic manipulation tasks. *Robotics and Computer-Integrated Manufacturing*, 81: 102517, 2023.

D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. Technical report, University of Montreal, 2009. Technical report.

J. Eschmann. Reward function design in reinforcement learning. *Reinforcement learning algorithms: Analysis and Applications*, pp. 25–33, 2021.

B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

B. Eysenbach, R. Salakhutdinov, and S. Levine. The information geometry of unsupervised reinforcement learning. *arXiv preprint arXiv:2110.02719*, 2021.

A. Fabisch, C. Petzoldt, M. Otto, and F. Kirchner. A survey of behavior learning applications in robotics–state of the art and perspectives. *arXiv preprint arXiv:1906.01868*, 2019.

R. Fakoor, P. Chaudhari, and A. J. Smola. P3o: Policy-on policy-off policy optimization. In *Uncertainty in artificial intelligence*, pp. 1017–1027. PMLR, 2020.

R. Featherstone. *Rigid body dynamics algorithms*. Springer, 2008.

R. Fernandez-Fernandez, M. Aggravi, R. P. Giordano, J. G. Victores, and C. Pacchierotti. Neural style transfer with twin-delayed ddpg for shared control of robotic manipulators. In *International Conference on Robotics and Automation (ICRA)*, pp. 4073–4079. IEEE, 2022.

N. Ferns, P. Panangaden, and D. Precup. Metrics for finite markov decision processes. In *UAI*, volume 4, pp. 162–169, 2004.

S. Filippi, O. Cappé, and A. Garivier. Optimism in reinforcement learning and kullback-leibler divergence. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 115–122. IEEE, 2010.

R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T.H. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V.Seguy, D. J. Sutherland, R.Tavenard, A. Tong, and T. Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021. URL `http://jmlr.org/papers/v22/20-451.html`.

D. J. Foster, S. M. Kakade, J. Qian, and A. Rakhlin. The statistical complexity of interactive decision making. *arXiv preprint arXiv:2112.13487*, 2021.

V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

T. Freiesleben and T. Grote. Beyond generalization: a theory of robustness in machine learning. *Synthese*, 202(4):109, 2023.

S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

S. Fujimoto, E. Conti, M. Ghavamzadeh, and J. Pineau. Benchmarking batch deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 2019.

H. Furuta, T. Matsushima, T. Kozuno, Y. Matsuo, S. Levine, O. Nachum, and S. S. Gu. Policy information capacity: Information-theoretic measure for task complexity in deep reinforcement learning. *Proceedings of the 38th International Conference on Machine Learning*, 139:3541–3552, 2021.

A. Ganguly and S. W. F. Earp. An introduction to variational inference. *arXiv preprint arXiv:2108.13083*, 2021.

Y. Gao and P. Chaudhari. An information-geometric distance on the space of tasks. In *Proceedings of the 38th International conference on machine learning*, pp. 3553–3563. PMLR, 2021.

P. Garnier, J. Viquerat, J. Rabault, A. Larcher, A. Kuhnle, and E. Hachem. A review on deep reinforcement learning for fluid mechanics. *Computers and Fluids*, 225: 104973, 2021.

C. Gelada, S. Kumar, J. Buckman, O. Nachum, and M. G. Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *Proceedings*

of the 36th International Conference on Machine Learning, pp. 2170–2179. PMLR, 2019.

A. Gepperth and B. Pfülb. Gradient-based training of gaussian mixture models for high-dimensional streaming data. *Neural Processing Letters*, 53(6):4331–4348, 2021.

A. Gibbs and F. E. Su. On choosing and bounding probability metrics. *International Statistical Review / Revue Internationale de Statistique*, 70(3):419–435, 2002.

R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in markov decision processes. *Artificial intelligence*, 147(1-2):163–223, 2003.

A. Givchi. *Optimal Transport in Reinforcement Learning*. PhD thesis, Graduate School-Newark Rutgers, The State University of New Jersey, 2021. URL `https://rucore.libraries.rutgers.edu/rutgers-lib/66700/PDF/1/`. Doctoral thesis.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*, volume 1. MIT press Cambridge, 2016.

A. Gorodetsky, S. Karaman, and Y. Marzouk. High-dimensional stochastic optimal control using continuous tensor decompositions. *The International Journal of Robotics Research*, 37(2-3):340–377, 2018.

J. Grothoff, N. C. Torres, and T. Kleinert. Assessment of reinforcement learning applications for industrial control based on complexity measures. *at-Automatisierungstechnik*, 70(1):53–66, 2022.

M. Grześ. Reward shaping in episodic reinforcement learning. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 565–573, 2017.

S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *International conference on machine learning*, pp. 2829–2838, 2016.

R. S. Guillén, L. V. Calderita, A. Hidalgo-Paniagua, and R. J. P. Bandera. Measuring smoothness as a factor for efficient and socially accepted robot motion. *Sensors*, 20 (23):6822, 2020.

K. Gurney. *An introduction to neural networks*. UCL Press Limited, 1997.

D. Ha and J. Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pp. 1352–1361. PMLR, 2017.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2019.

D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models. *nature*, 2025. URL `https://doi.org/10.1038/s41586-025-08744-2`.

D. Han, B. Mulyana, V. Stankovic, and S. Cheng. A survey on deep reinforcement learning algorithms for robotic manipulation. *Sensors*, 23(7):3762, 2023.

M. Han, L. Zhang, J. Wang, and W. Pan. Actor-critic reinforcement learning for control with stability guarantee. *IEEE Robotics and Automation Letters*, 5(4):6217–6224, 2020.

T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning: data mining, inference, and prediction, 2009.

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, number 392, pp. 3207–3214, 2018.

A. Hentout, A. Maoudj, and M. Aouache. A review of the literature on fuzzy-logic approaches for collision-free path planning of manipulator robots. *Artificial Intelligence Review*, 56(4):3369–3444, 2023.

O. Hernández-Lerma and J. B. Lasserre. *Discrete-time Markov control processes: basic optimality criteria*. Springer New York, 1996.

I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*, 2017a.

I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *International conference on machine learning*, pp. 1480–1490. PMLR, 2017b.

L. Hofer and H. Gimbert. Online reinforcement learning for real-time exploration in continuous state and action markov decision processes. *arXiv preprint arXiv:1612.03780*, 2016.

G. A. Holton. *Value-at-risk: theory and practice*. Academic Press, 2003. URL `https://first-edition.value-at-risk.net/wp-content/uploads/book_pdf/Holton-2003-Value-at-Risk.pdf`.

L. Hou, L. Pang, X. Hong, Y. Lan, Z. Ma, and D. Yin. Robust reinforcement learning with wasserstein constraint. *arXiv preprint arXiv:2006.00945*, 2020.

Y. Hou, H. Hong, Z. Sun, D. Xu, and Z. Zeng. The control method of twin delayed deep deterministic policy gradient with rebirth mechanism to multi-dof manipulator. *Electronics*, 10(7):870, 2021.

R. A. Howard. *Dynamic programming and markov processes.* MIT Press and John Wiley, 1960.

J. Huang, X. Su, Z. Fang, and H. Kasai. Anchor space optimal transport as a fast solution to multiple optimal transport problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

Q. Huang. Model-based or model-free, a review of approaches in reinforcement learning. In *2020 International Conference on Computing and Data Science (CDS)*, pp. 219–221, 2020.

A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys*, 50:1—-35, 2017.

M. Hutsebaut-Buysse, K. Mets, and S. Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221, 2022.

J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, 2021.

S. Ibrahim, M. Mostafa, A. Jnadi, H. Salloum, and P. Osinenko. Comprehensive overview of reward engineering and shaping in advancing reinforcement learning applications. *IEEE Access*, 2024.

R. Islam, P. Henderson, M. Gomrokchi, and D. Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, 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*, 364(6443):859–865, 2019.

T. Jaksch, R. Ortner, and P. Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(51):1563–1600, 2010. URL `http://jmlr.org/papers/v11/jaksch10a.html`.

N. Jiang and A. Agarwal. Open problem: The dependence of sample complexity lower bounds on planning horizon. In *Conference On Learning Theory*, pp. 3395–3398. PMLR, 2018.

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

S. M. Jordan, A. White, B. C. da Silva, M. White, and P. S. Thomas. Position: Benchmarking is limited in reinforcement learning research. In *International conference on machine learning (ICML)*, 2024.

N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv preprint arXiv:1806.10729*, 2018.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Journal of artificial intelligence research. *Reinforcement Learning: A Survey*, 4:237—-285, 1996.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, 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. *arXiv preprint arXiv:1903.00374*, 2019.

S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 267–274. Morgan Kaufmann Publishers Inc., 2002.

S. M. Kakade. *On the sample complexity of reinforcement learning.* PhD thesis, Gatsby Computational Neuroscience Unit, University College London, United Kingdom, March 2003. Doctoral thesis.

K. Kalagarla, R. Jain, and P. Nuzzo. A sample-efficient algorithm for episodic finite-horizon mdp with constraints. In *The 35th AAAI Conference on Artificial Intelligence*, pp. 8030–8037, 2021.

M. Kallel, D. Basu, R. Akrour, and C. D'Eramo. Augmented bayesian policy search. In *The 12th International Conference of Learning Representations*, volume 139, 2024.

N. Kallus and M. Uehara. Efficiently breaking the curse of horizon in off-policy evaluation with double reinforcement learning. *Operations Research*, 70(6):3282–3302, 2022.

N. Kantas. *Sequential decision making in general state space models.* PhD thesis, Department of Engineering, University of Cambridge, United Kingdom, March 2009. Doctoral thesis.

J. Karwowski, O. Hayman, X. Bai, K. Kiendlhofer, C. Griffin, and J. M. V. Skalse. Goodhart's law in reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=5o9G4XF1LI`.

E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976): 982–987, 2023.

K. Kawaguchi, Z. Deng, K. Luh, and J. Huang. Robustness implies generalization via data-dependent generalization bounds. In *International conference on machine learning*, pp. 10866–10894. PMLR, 2022.

M. Khadem, L. Da Cruz, and C. Bergeles. Force/velocity manipulability analysis for 3d continuum robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4920–4926. IEEE, 2018.

S. A. Khader, H. Yin, P. Falco, and D. Kragic. Learning stable normalizing-flow control for robotic manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1644–1650. IEEE, 2021.

D. Kingma, T. Salimans, B. Poole, and J. Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2017.

R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013. URL `https://doi.org/10.1177/0278364913495721`.

D. E. Koditschek. What is robotics? why do we need it and how can we get it? *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1):1–33, 2021.

J. Köhler, A. Krämer, and F. Noé. Smooth normalizing flows. *Advances in Neural Information Processing Systems*, 34:2796–2809, 2021.

S. Kolouri, S. R. Park, M. Thorpe, D. Slepcev, and G. K. Rohde. Optimal mass transport: Signal processing and machine-learning applications. *IEEE Signal Processing Magazine*, 34:43–59, 2017.

S. Kolouri, K. Nadjahi, U. Simsekli, R. Badeau, and G. Rohde. Generalized sliced wasserstein distances. *Advances in neural information processing systems*, 32, 2019.

K. Konakli and B. Sudret. Global sensitivity analysis using low-rank tensor approximations. *Reliability Engineering & System Safety*, 156:64–83, 2016.

P. Kormushev, S. Calinon, and D. G. Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

O. Kroemer, S. Niekum, and G. Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *Journal of machine learning research*, 22(30):1–82, 2021.

S. Kulkarni and D. D. Patil. Reinforcement learning for autonomous systems. In *2025 4th International Conference on Sentiment Analysis and Deep Learning (ICSADL)*, pp. 816–820. IEEE, 2025.

V. Kunc and J. Kléma. Three decades of activations: A comprehensive survey of 400 activation functions for neural networks. *arXiv preprint arXiv:2402.09092*, 2024.

I. Kuzborskij and C. Lampert. Data-dependent stability of stochastic gradient descent. In *international conference on machine learning (ICML)*, pp. 2815–2824. PMLR, 2018.

P. Ladosz, L. Weng, M. Kim, and H. Oh. Exploration in deep reinforcement learning: A survey. *Information Fusion*, 85:1–22, 2022.

C. Laidlaw, S. Russell, and A. Dragan. Bridging rl theory and practice with the effective horizon. *Advances in Neural Information Processing Systems*, 36:58953–59007, 2023.

S. Lange and M. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 international joint conference on neural networks (IJCNN)*, pp. 1–8. IEEE, 2010.

R. Laroche and R. T. des Combes. On the occupancy measure of non-markovian policies in continuous mdps. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, 2023.

R. Laroche, R. T. des Combes, and J. Buckman. Non-markovian policies occupancy measures. *arXiv preprint arXiv:2205.13950*, 2022.

M. Laskin, H. Liu, X. Bin Peng, D. Yarats, A. Rajeswaran, and P. Abbeel. CIC: contrastive intrinsic control for unsupervised skill discovery. *CoRR*, abs/2202.00161, 2022. URL `https://arxiv.org/abs/2202.00161`.

T. Lattimore and C. Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

A. Lazaridis, A. Fachantidis, and I. Vlahavas. Deep reinforcement learning: A state-of-the-art walkthrough. *Journal of Artificial Intelligence Research*, 69:1421–1471, 2020. URL `https://doi.org/10.1613/jair.1.12412`.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

J. Lee, M. Dabagia, E. Dyer, and C. Rozell. Hierarchical optimal transport for multimodal distribution alignment. *Advances in Neural Information Processing Systems*, 32, 2019.

J. M. Lee. *Manifolds and differential geometry*, volume 107. American Mathematical Society, 2009.

A. C. Li, P. Vaezipoor, R. T. Icarte, and S. A. McIlraith. Challenges to solving combinatorially hard long-horizon deep rl tasks. *arXiv preprint arXiv:2206.01812*, 2022.

S. E. Li. Deep reinforcement learning. In *Reinforcement learning for sequential decision and optimal control*, pp. 365–402. Springer, 2023.

A. Likmeta, M. Sacco, A. M. Metelli, and M. Restelli. Wasserstein actor-critic: directed exploration via optimism for continuous-actions control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 8782–8790, 2023.

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. In *International Conference on Learning Representations*, 2016. URL `https://arxiv.org/pdf/1509.02971v6`.

Y. Lin, Y. Liu, F. Lin, L. Zou, P. Wu, W. Zeng, H. Chen, and C. Miao. A survey on reinforcement learning for recommender systems. *IEEE Transactions on Neural Networks and Learning Systems*, 35(10):13164–13184, 2023.

Q. Liu, L. Li, Z. Tang, and D. Zhou. Breaking the curse of horizon: Infinite-horizon off-policy estimation. *Advances in neural information processing systems*, 31, 2018.

R. Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Dresp-Langley. Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review. *Robotics*, 10(1):22, 2021.

S. Liu. An evaluation of ddpg, td3, sac, and ppo: Deep reinforcement learning algorithms for controlling continuous system. In *International Conference on Data Science, Advanced Algorithm and Intelligent Computing*, 2023.

Y. Liu, M. Yang, Z. Zhang, F. Bao, Y. Cao, and G. Zhang. Diffusion-model-assisted supervised learning of generative models for density estimation. *Journal of Machine Learning for Modeling and Computing*, 5(1), 2024.

A. Lobbezoo, Q. Yanjun, and K. Hyock-Ju. Reinforcement learning for pick and place operations in robotics: A survey. *Robotics*, 10(3), 2021.

N. G. Lopez, Y. L. E. Nuin, E. B. Moral, L. U. S. Juan, A. S. R., V. M. Vilches, and R. Kojcev. gym-gazebo2, a toolkit for reinforcement learning using ros 2 and gazebo. *arXiv preprint arXiv:1903.06278*, 2019.

J. Lott. Some geometric calculations on wasserstein space. *Communications in Mathematical Physics*, 277:423—-437, 2008.

C. Lv, G. Chen, H. Zhao, J. Chen, and H. Yu. An adaptive robust hybrid force/position control for robot manipulators system subject to mismatched and matched disturbances. *IEEE Access*, 12:42264–42278, 2024.

P. MacAlpine and P. Stone. Overlapping layered learning. *Artificial Intelligence*, 254: 21–43, 2018.

G. D. Maggiora, L. A. Croquevielle, N. Deshpande, H. Horsley, T. Heinis, and A. Yakimovich. Conditional variational diffusion models. In *The Twelfth International Conference on Learning Representations*, 2023.

R. A. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra. Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on robot learning*, pp. 561–591, 2018.

J. A. Marshall, A. Bonchis, E. Nebot, and S. Scheding. Robotics in mining. In *Springer handbook of robotics*, pp. 1549–1576. Springer, 2016.

M. T. Mason. Toward robotic manipulation. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):1–28, 2018.

M. J. Mataric. Reward functions for accelerated learning. In *Machine Learning Proceedings 1994*, pp. 181–189. Elsevier, 1994. ISBN 978-1-55860-335-6. doi: https://doi.org/10.1016/B978-1-55860-335-6.50030-1. URL `https://www.sciencedirect.com/science/article/pii/B9781558603356500301`.

G. Matheron, N. Perrin, and O. Sigaud. The problem with DDPG: understanding failures in deterministic environments with sparse rewards. *arXiv preprint arXiv:1911.11679*, 2019.

M. J. Mathew. A self model learning robotic manipulator leading to eye-hand coordination, 2014. Master of Technology thesis.

K. Mehlhorn, B. R. Newell, P. M. Todd, M. D. Lee, K. Morgan, V. A. Braithwaite, D. Hausmann, K. Fiedler, and C. Gonzalez. Unpacking the exploration-exploitation tradeoff: a synthesis of human and animal literatures. *Decision*, 2(3):191, 2015.

M. Memmel, P. Liu, D. Tateo, and J. Peters. Dimensionality reduction and prioritized exploration for polic search. In *25th International Conference on Artificial Intelligence and Statistics*, 2022.

A. M. Metelli, A. Likmeta, and M. Restelli. Propagating uncertainty in reinforcement learning via wasserstein barycenters. *Advances in Neural Information Processing Systems*, 32, 2019.

C. D. Meyer. The role of the group generalized inverse in the theory of finite markov chains. *SIAM Review*, 17(3):443–464, 1975.

U. Michelucci. An introduction to autoencoders. *arXiv preprint arXiv:2201.03898*, 2022.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Reidmiller, A. K. Fidjeland, G. Ostrovski, S. Peterson, 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*, pp. 529—533, 2015.

T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker. Model-based reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 16(1):1–118, 2023.

A. Mohtasib, G. Neumann, and H. Cuayáhuitl. A study on dense and sparse (visual) rewards in robot policy learning. In *Annual Conference Towards Autonomous Robotic Systems*, pp. 3–13. Springer, 2021.

A. W. Moore. *Efficient Memory-based Learning for Robot Control*. PhD thesis, University of Cambridge, 1990. Doctoral thesis.

J. Moos, K. Hansel, H. Abdulsamad, S. Stark, D. Clever, and J. Peters. Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction*, 4(1):276–315, 2022.

E. F. Morales, R. Murrieta-Cid, I. Becerra, and M. A. Esquivel-Basaldua. A survey on deep learning and deep reinforcement learning in robotics with a tutorial on deep reinforcement learning. *Intelligent Service Robotics*, 14(5):773–805, 2021.

S. Müller, A. von Rohr, and S. Trimpe. Local policy search with bayesian optimization. *Advances in Neural Information Processing Systems*, 34:20708–20720, 2021.

K. P. Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, 2012. ISBN 9780262018029.

R. M. Murray, Z. Li, and S. S. Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

V. Myers, C. Ji, and B. Eysenbach. Horizon generalization in reinforcement learning. *arXiv preprint arXiv:2501.02709*, 2025.

O. Nachum, H. Tang, X. Lu, S. Gu, H. Lee, and S. Levine. Why does hierarchy (sometimes) work so well in reinforcement learning? *arXiv preprint arXiv:1909.10618*, 2019.

J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal. Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6):737–757, 2008.

S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020.

S. S. Narvekar. *Curriculum learning in reinforcement learning*. PhD thesis, The University of Texas at Austin, 2021. Order Number: 29605085.

G. Neu and C. Pike-Burke. A unifying view of optimism in episodic reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1392–1403, 2020.

K. Nguyen. An introduction to sliced optimal transport. *arXiv preprint arXiv:2508.12519*, 2025.

A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*, 2018.

D. Nikovski. Reinforcement learning in continuous spaces, 1998. URL `https://www.cs.cmu.edu/~rll/overview/danieln_01/`. Website.

R. M. Nkhumise, D. Basu, T. J. Prescott, and A. Gilra. Studying exploration in rl: An optimal transport analysis of occupancy measure trajectories. *Transactions on Machine Learning Research (TMLR)*, 2025.

R. M. Nkhumise, M. S. Talamali, and A. Gilra. Issues with measuring task complexity via random policies in robotic tasks. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2026. To appear.

J. Obando-Ceron, J. G. M. Araújo, A. Courville, and P. S. Castro. On the consistency of hyper-parameter selection in value-based deep reinforcement learning. In *Reinforcement Learning Conference (RLC)*. Reinforcement Learning Journal (RLJ), 2024.

D. Oller, T. Glasmachers, and G. Cuccu. Analyzing reinforcement learning benchmarks with random weight guessing. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 975–982, 2020.

M. M. J. Opgenoord, D. L. Allaire, and K. E. Willcox. Variance-based sensitivity analysis to support simulation-based design under uncertainty. *Journal of Mechanical Design*, 138(11):111410, 2016.

I. Osband and B. Van Roy. Why is posterior sampling better than optimism for reinforcement learning? In *International conference on machine learning*, pp. 2701–2710. PMLR, 2017.

I. Osband, B. V. Roy, and D. Russo. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, pp. 3003—-3011, 2013.

I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepesvari, S. Singh, B. van Roy, R. Sutton, D. Silver, and H. van Hassel. Behaviour suite for reinforcement learning. *arXiv preprint arXiv:1908.03568*, 2019.

R. Ouhamma, D. Basu, and O. Maillard. Bilinear exponential family of mdps: frequentist regret bound with tractable exploration & planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 9336–9344, 2023.

A. Pacchiano, P. Ball, J. Parker-Holder, K. Choromanski, and S. Roberts. Towards tractable optimism in model-based reinforcement learning. In *Uncertainty in Artificial Intelligence*, pp. 1413–1423. PMLR, 2021.

A. Pagnoni, K. Liu, and S. Li. Conditional variational autoencoder for neural machine translation. *arXiv preprint arXiv:1812.04405*, 2018.

K. Panaganti, Z. Xu, D. Kalathil, and M. Ghavamzadeh. Robust reinforcement learning using offline data. *Advances in neural information processing systems*, 35:32211–32224, 2022.

V. M. Panaretos and Y. Zemel. Statistical aspects of wasserstein distances. *Annual Review of Statistics and its Applications*, 6:405–431, 2019. URL `https://doi.org/10.1146/annurev-statistics-030718-104938`.

G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

S. Park, K. Frans, D. Mann, B. Eysenbach, A. Kumar, and S. Levine. Horizon reduction makes rl scalable. In *Advances in Neural Information Processing Systems*, 2025.

Paryanto, M. Brossog, M. Bornschlegl, and J. Franke. Reducing the energy consumption of industrial robots in manufacturing systems. *The International Journal of Advanced Manufacturing Technology*, 78(5):1315–1328, 2015.

M. Paschali, S. Conjeti, F. Navarro, and N. Navab. Generalizability vs. robustness: adversarial examples for medical imaging. *arXiv preprint arXiv:1804.00504*, 2018.

D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

A. Patterson, S. Neumann, M. White, and A. White. Empirical design in reinforcement learning. *Journal of Machine Learning Research*, 25(318):1–63, 2024.

A. Perrusquía, J. A. Flores-Campos, and C. R. Torres-San-Miguel. A novel tuning method of pd with gravity compensation controller for robot manipulators. *IEEE Access*, 8:114773–114783, 2020.

V. Petrone, E. Ferrentino, and P. Chiacchio. The dynamic model of the ur10 robot and its ros2 integration. *IEEE Transactions on Industrial Informatics*, 2025.

G. Peyré. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

E. Pignatelli, J. Ferret, M. Geist, T. Mesnard, H. van Hasselt, O. Pietquin, and L. Toni. A survey of temporal credit assignment in deep reinforcement learning. *Transactions on Machine Learning Research (TMLR)*, 2023.

M. Pirotta, M. Restelli, and L. Bascetta. Policy gradient in lipschitz markov decision processes. *Machine Learning*, 100(2-3):255—283, 2015.

M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

A. S. Polydoros and L. Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

M. L. Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

M. L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley and Sons, 1994.

S. Raghvendra, P. Shirzadian, and K. Zhang. A new robust partial p-wasserstein-based metric for comparing distributions. *arXiv preprint arXiv:2405.03664*, 2024.

R. Rajan, J. L. B. Diaz, S. Guttikonda, F. Ferreira, A. Biedenkapp, J. O. von Hartz, and F. Hutter. Mdp playground: An analysis and debug testbed for reinforcement learning. *Journal of Artificial Intelligence Research*, 77:821–890, 2023.

A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade. Towards generalization and simplicity in continuous control. *Advances in neural information processing systems*, 30, 2017.

G. O. Ramos, B. C. da Silva, and A. L. C. Bazzan. Learning to minimise regret in route choice. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pp. 846–855, 2017.

G. O. Ramos, A. L. C. Bazzan, and B. C. da Silva. Analysing the impact of travel information for minimising the regret of route choice. *Transportation Research Part C: Emerging Technologies*, 88:257–271, 2018.

K. Rathbone. *Evolving visually guided neural network robot arm controllers for lifetime learning.* PhD thesis, University of Sheffield, 2000. Doctoral thesis.

Z. Ren. The advance of generative model and variational autoencoder. In *2022 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS)*, pp. 268–271. IEEE, 2022.

D. Reynolds. *Gaussian Mixture Models*. Springer, 2009.

R. Rocchetta, L. Bellani, M. Compare, E. Zio, and E. Patelli. A reinforcement learning framework for optimal operation and maintenance of power grids. *Applied energy*, 241:291–301, 2019.

A. Rosenfeld, M. Cohen, M. E. Taylor, and S. Kraus. Leveraging human knowledge in tabular reinforcement learning: A study of human subjects. *The knowledge engineering review*, 33:e14, 2018.

Y. P. Ruan and Z. H. Ling. Emotion-regularized conditional variational autoencoder for emotional response generation. *IEEE Transactions on Affective Computing*, 14 (1):842–848, 2021.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*, pp. 318–362. The MIT press, 1986.

S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010. ISBN 9780136042594.

S. K. Saha. *Introduction to robotics*. Tata McGraw-Hill Education, 2014.

H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.

A. Saltelli and P. Annoni. How to avoid a perfunctory sensitivity analysis. *Environmental Modelling & Software*, 25(12):1508–1517, 2010.

A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto. *Sensitivity Analysis in Practice: a guide to assessing scientific models*. John Wiley Sons, 2004. ISBN 0470870931.

A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola. Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index. *Computer physics communications*, 181(2):259–270, 2010.

S. L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data mining and knowledge discovery*, 1(3):317–328, 1997.

F. Santambrogio. *Optimal Transport for Applied Mathematicians*. Bikhauser Cham, 2015. ISBN 978-3-319-20828-2.

M. J. Schervish, J. B. Kadane, and T. Seidenfeld. Characterization of proper and strictly proper scoring rules for quantiles. *Preprint, Carnegie Mellon University*, 18, 2012. URL `https://www.cmu.edu/dietrich/philosophy/docs/seidenfeld/Scoring%20Rules%20for%20Quantiles.pdf`.

J. Schmidhuber, S. Hochreiter, and Y. Bengio. Evaluating benchmark problems by random guessing. In *A Field Guide to Dynamical Recurrent Networks*, pp. 1329– –1338. Wiley, 1999.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

J. Schulman, X. Chen, and P. Abbeel. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017a.

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

L. Sciavicco and B. Siciliano. *Modelling and control of robot manipulators*. Springer, 2012.

H. Van Seijen, H. Nekoei, E. Racah, and S. Chandar. The loca regret: a consistent metric to evaluate model-based behavior in reinforcement learning. *Advances in Neural Information Processing Systems*, 33:6562–6572, 2020.

A. A. Shahid, D. Piga, F. Braghin, and L. Roveda. Continuous control actions learning and adaptation for robotic manipulation through reinforcement learning. *Autonomous Robots*, 46(3):483–498, 2022.

S. Shalev-Shwartz, O. Shamir, N. Srebro, and K. Sridharan. Learnability and stability in the general learning setting. In *COLT*, 2009.

B. Siciliano, O. Khatib, and T. Kröger. *Springer handbook of robotics*, volume 200. Springer, 2008.

B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: modelling, planning and control*. Springer, 2009.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning (ICML)*, pp. 387–395, 2014.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

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*, 362(6419):1140–1144, 2018.

S. R. Sinclair, S. Banerjee, and C. L. Yu. Adaptive discretization in online reinforcement learning. *Operations Research*, 71(5):1636–1652, 2023.

N. S. Sandeepand V. Sinha and K. P. Pradyumna. Experimental evaluation of reinforcement learning algorithms. In *International Conference on Computational Intelligence and Data Engineering*, pp. 469–484. Springer, 2022.

I. M. Sobol. Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and computers in simulation*, 55(1-3):271–280, 2001.

K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

M. Sommerfeld, J. Schrieber, Y. Zemel, and A. Munk. Optimal transport: Fast probabilistic approximation with exact solvers. *Journal of Machine Learning Research*, 20:1–23, 2019.

X. Song, Y., and J. Jackson. An empirical study on hyperparameters and their interdependence for rl generalization. *ICML 2019 Workshop on Understanding and Improving Generalization in Deep Learning*, 2019. URL `arXivpreprintarXiv:1906.00431`.

S. P. K. Spielberg, R. B. Gopaluni, and P. D. Loewen. Deep reinforcement learning approaches for process control. In *2017 6th international symposium on advanced control of industrial processes (AdCONIP)*, pp. 201–206. IEEE, 2017.

M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot modeling and control*, volume 3. John Wiley & Sons, 2006.

M. Strens. A bayesian framework for reinforcement learning. In *International conference on machine learning (ICML)*, volume 2000, pp. 943–950, 2000.

F. Stulp and S. Schaal. Hierarchical reinforcement learning with movement primitives. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pp. 231–238. IEEE, 2011.

K. Sun, Y. Zhao, Y. Liu, B. Jiang, and L. Kong. Distributional reinforcement learning via sinkhorn iterations. *arXiv preprint arXiv:2202.00769*, 2022.

W. Sun. *Stability of machine learning algorithms*. PhD thesis, Purdue University, May 2015. Doctoral thesis.

N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke. The limits and potentials of deep learning for robotics. *The International journal of robotics research*, 37(4-5):405–420, 2018.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018.

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*, 12, 1999a.

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999b.

R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 761–768, 2011.

U. Syed, M. Bowling, and R. E. Schapire. Apprenticeship learning using linear programming. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 1032–1039. PMLR, 2008.

J. L. Tan, B. A. Taha, N. A. Aziz, M. H. H. Mokhtar, M. Mukhlisin, and N. Arsad. A review of reinforcement learning evolution: Taxonomy, challenges and emerging solutions. *International Journal of Advanced Computer Science & Applications*, 16 (1), 2025.

C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone. Deep reinforcement learning for robotics: A survey of real-world successes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 28694–28698, 2025.

H. Tang, R. Houthooft, D. Foote, X. Chen A. Stooke, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2753–2762. Curran Associates, Inc., 2017.

Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancqm, T. Lillicrap, and M. Reidmiller. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

J. Terven. Deep reinforcement learning: a chronological overview and methods. *AI*, 6 (3):46, 2025.

G. Thomas, M. Chien, A. Tamar, J. A. Ojea, and P. Abbeel. Learning robotic assembly from cad. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3524–3531, 2018.

S. Thrun. Efficient exploration in reinforcement learning. Technical report, Carnegie Mellon University, 1992. Technical report.

S. Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 connectionist models summer school*, pp. 255–263. Psychology Press, 2014.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623. URL `https://docs.ufpr.br/~danielsantos/ProbabilisticRobotics.pdf`.

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*, pp. 5026–5033. IEEE, 2012.

M. Tosin, A. M. A. Côrtes, and A. Cunha. A tutorial on sobol'global sensitivity analysis applied to biological models. *Networks in Systems Biology: Applications for Disease Modeling*, pp. 93–118, 2020.

M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, A. Pierré R. Perez-Vicente, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

D. Tran, J. Liu, M. W. Dusenberry, D. Phan, M. Collier, J. Ren, K. Han, Z. Wang, Z. Mariet, H. Hu, N. Band, T. G. J. Rudner, K. Singhal, Z. Nado, J. van Amersfoort, A. Kirsch, R. Jenatton, N. Thain, H. Yuan, K. Buchanan, K. Murphy, D. Sculley, Y. Gal, Z. Ghahramani, J. Snoek, and B. Lakshminarayanan. Plex: Towards reliability using pretrained large model extensions. *arXiv preprint arXiv:2207.07411*, 2022.

Vijay V. Konda and J. Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

N. Vahrenkamp, T. Asfour, G. Metta, G. Sandini, and R. Dillmann. Manipulability analysis. In *12th IEEE-RAS international conference on humanoid robots (humanoids 2012)*, pp. 568–573. IEEE, 2012.

D. Valencia, J. Jia, R. Li, A. Hayashi, M. Lecchi, R. Terezakis, T. Gee, M. V. Liarokapis, B. A. MacDonald, and H. Williams. Comparison of model-based and model-free reinforcement learning for real-world dexterous robotic manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 871–878, 2023.

P. Vamplew, B. J. Smith, J. Källström, G. Ramos, R. Rădulescu, D. M. Roijers, C. F. Hayes, F. Heintz, P. Mannion, P. J. K. Libin, R. Dazeley, and C. Foale. Scalar reward is not enough: a response to silver, singh, precup and sutton (2021). *Autonomous Agents and Multi-Agent Systems*, 36(2):41, 2022.

H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

R. A. Vandermeulen, W. M. Tai, and B. Aragam. Breaking the curse of dimensionality in structured density estimation. *Advances in Neural Information Processing Systems*, 37:65852–65883, 2024.

C. Villani. *Optimal transport, old and new*, volume 338. Springer, 2008.

P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

C. A. Voelcker, M. Hussing, and E. Eaton. Can we hop in general? a discussion of benchmark selection and design using the hopper environment. *arXiv preprint arXiv:2410.08870*, 2024.

A. Wagenmaker, Z. Zhou, and S. Levine. Behavioral exploration: Learning to explore via in-context adaptation. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

H. Wang, A. Sakhadeo, A. White, J. Bell, V. Liu, X. Zhao, P. Liu, T. Kozuno, A. Fyshe, and M. White. No more pesky hyperparameters: Offline hyperparameter tuning for rl. *arXiv preprint arXiv:2205.08716*, 2022.

J. Wang, Y. Liu, and B. Li. Reinforcement learning with perturbed rewards. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 6202–6209, 2020.

J. Wang, I. Trummer, and D. Basu. Udo: universal database optimization using reinforcement learning. *Proceedings of the VLDB Endowment*, 14(13):3402–3414, 2021.

T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

X. Wang, S. Wang, X. Liang, D. Zhao, J. Huang, X. Xu, B. Dai, and Q. Miao. Deep reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4):5064–5078, 2024.

P. N. Ward. Linear programming in reinforcement learning, 2021. URL `https://escholarship.mcgill.ca/downloads/xs55mh725`. MSc thesis.

C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

J. Weng, H. Chen, D. Yan, K. You, A. Duburcq, M. Zhang, Y. Su, H. Su, and J. Zhu. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 23(267):1–6, 2022.

R. M. West. Best practice in statistics: Use the welch t-test when testing the difference between two groups. *Annals of clinical biochemistry*, 58(4):267–269, 2021.

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)*, pp. 120–127. IEEE, 2011.

B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 2002.

T. T. Wong and P. Y. Yeh. Reliable accuracy estimates from k-fold cross validation. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1586–1594, 2020.

C. Wu, T. Li, Z. Zhang, and Y. Yu. Bayesian optimistic optimization: Optimistic exploration for model-based reinforcement learning. *Advances in neural information processing systems*, 35:14210–14223, 2022.

Y. Wu and J. Feng. Development and application of artificial neural network. *Wireless Personal Communications*, 102(2):1645–1656, 2018.

Y. Wu and Y. Tian. Training agent for first-person shooter game with actor-critic curriculum learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Y. Xie, X. Wang, R. Wang, and H. Zha. A fast proximal point method for computing exact wasserstein distance. In *Uncertainty in artificial intelligence*, pp. 433–453. PMLR, 2020.

H. Xu and S. Mannor. Robustness and generalization. *Machine learning*, 86(3):391–423, 2012.

J. Xu, Y. Tian, P. Ma, D. Rus, S. Sueda, and W. Matusik. Prediction-guided multiobjective reinforcement learning for continuous robot control. In *International conference on machine learning*, pp. 10607–10616. PMLR, 2020.

Z. Xu and T. Campbell. Embracing the chaos: analysis and diagnosis of numerical instability in variational flows. *Advances in Neural Information Processing Systems*, 36:32360–32386, 2023.

C. Yang, Y. Xie, S. Liu, and D. Sun. Force modeling, identification, and feedback control of robot-assisted needle insertion: a survey of the literature. *Sensors*, 18(2): 561, 2018.

T. Yang, L. Zhao, W. Li, and A. Y. Zomaya. Reinforcement learning in sustainable energy and electric systems: A survey. *Annual Reviews in Control*, 49:145–163, 2020.

Y. Yang, T. Zhou, Q. He, L. Han, M. Pechenizkiy, and M. Fang. Task adaptation from skills: Information geometry, disentanglement, and new objectives for unsupervised reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024.

D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus. Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the aaai conference on artificial intelligence*, volume 35, pp. 10674–10681, 2021.

H. Yau. *On the interpretability of reinforcement learning*. PhD thesis, Faculty of Engineering and Physical Sciences, University of Surrey, United Kingdom, March 2024. Doctoral thesis.

W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao. Mastering atari games with limited data. *Advances in neural information processing systems*, 34:25476–25488, 2021.

T. Yoshikawa. Manipulability of robotic mechanisms. *The international journal of Robotics Research*, 4(2):3–9, 1985.

X. Yuan, Y. Wang, R. Zhang, Q. Gao, Z. Zhou, R. Zhou, and F. Yin. Reinforcement learning control of hydraulic servo system based on td3 algorithm. *Machines*, 10(12): 1244, 2022.

Y. Yuan, Z. L. Yu, L. Hua, Y. Cheng, J. Li, and X. Sang. Hierarchical dynamic movement primitive for the smooth movement of robots based on deep reinforcement learning. *Applied Intelligence*, 53(2):1417–1434, 2023.

C. Zhang, R. Barbano, and B. Jin. Conditional variational autoencoder for learned image reconstruction. *Computation*, 9(11):114, 2021a.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021b.

H. Zhang and T. Yu. Taxonomy of reinforcement learning algorithms. In *Deep reinforcement learning: Fundamentals, research and applications*, pp. 125–133. Springer, 2020.

L. Zhang, K. Tang, and X. Yao. Explicit planning for efficient exploration in reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32, pp. 7488–7497. Curran Associates, Inc., 2019.

M. Zhang, Y. Wang, X. Ma, L. Xia, J. Yang, Z. Li, and X. Li. Wasserstein distance guided adversarial imitation learning with reward shape exploration. In *IEEE 9th Data Driven Control and Learning Systems Conference*, pp. 1165–1170, 2020.

T. Zhang and H. Mo. Reinforcement learning for robot research: A comprehensive review and open issues. *International Journal of Advanced Robotic Systems*, 18(3): 17298814211007305, 2021.

Y. Zhang, L. Ke, A. Deshpande, A. Gupta, and S. Srinivasa. Cherry-picking with reinforcement learning. In *Robotics: Science and Systems*, 2023.

C. Zhu, X. Wang, T. Han, S. S. Du, and A. Gupta. Transferable reinforcement learning via generalized occupancy models. *arXiv preprint arXiv:2403.06328*, 2024.

Appendices

Appendix A

Theoretical Analysis of Exploration

This section contains additional proofs of proposition introduced in Chapter 3.

A.1 Proof of Proposition 1

The Linear Programming formulation for solving MDPs, assuming discrete state and action spaces, is (Puterman, 1994):

$$\begin{aligned} &\max_{v_\pi} \sum_{s,a} r(s,a) v_\pi(s,a) \\ &\text{subject to} \sum_a v_\pi(s,a) = p_0(s) + \gamma \sum_{s',a} T(s \mid s',a) v_\pi(s',a) \\ &\qquad v_\pi(s,a) \geq 0 \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A}\,, \end{aligned} \tag{A.1}$$

where $p_0(s)$ is the initial state distribution and $T(s \mid s', a)$ is the transition probability. The constraints of this optimization problem are often referred to as *Bellman Flow Constraint*. Note that this formulation naturally extends to continuous state and action spaces (where summations are replaced by integrals) without making additional assumptions (Puterman, 1994; Hernández-Lerma & Lasserre, 1996). Nevertheless, for clarity of exposition, we present Equation A.1 and subsequent equations in this section in the discrete setting to illustrate the proof.

A stationary policy π has a corresponding occupancy measure $v_\pi(s,a)$ that satisfies the Bellman flow constraint (Syed et al., 2008), and hence π and $v_\pi(s,a)$ share a bijective relationship (Syed et al., 2008; Givchi, 2021),

$$\pi(a \mid s) = \frac{v_\pi(s,a)}{u_\pi(s)} \tag{A.2}$$

with

$$u_\pi(s) = \sum_{a'} v_\pi(s,a') = p_0(s) + \gamma \sum_{s',a'} T(s \mid s',a') v_\pi(s',a') \tag{A.3}$$

By rearranging Equation A.2 to

$$v_\pi(s,a) = \pi(a \mid s) u_\pi(s) \tag{A.4}$$

and substituting Equation A.4 into Equation A.3, we can rewrite Equation A.3 as (defining $\mathcal{P}^\pi \triangleq \sum_a T(s \mid s', a)\pi(a \mid s')$),

$$\begin{aligned} p_0(s) &= u_\pi(s) - \gamma \sum_{s',a} T(s \mid s', a)\pi(a \mid s')u_\pi(s') \\ &\triangleq u_\pi(s) - \gamma \sum_{s'} \mathcal{P}^\pi(s \mid s')u_\pi(s') \end{aligned} \tag{A.5}$$

which in matrix form is

$$\begin{aligned} \mathbf{p}_0 &= \mathbf{u}_\pi - \gamma \mathbf{P}^\pi \mathbf{u}_\pi \\ &= \left(\mathbb{I} - \gamma \mathbf{P}^\pi\right) \mathbf{u}_\pi \,, \end{aligned} \tag{A.6}$$

where $\mathbf{p}_0, \mathbf{u}_\pi \in \mathbb{R}^{|\mathcal{S}|}$ are column vectors and $\mathbf{P}^\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ are matrices. Solving for $\mathbf{u}_\pi$, we get

$$\mathbf{u}_\pi = \left(\mathbb{I} - \gamma \mathbf{P}^\pi\right)^{-1} \mathbf{p}_0 \tag{A.7}$$

The inverse matrix $\left(\mathbb{I} - \gamma \mathbf{P}^\pi\right)^{-1}$ exists because for $\gamma < 1$, $\left(\mathbb{I} - \gamma \mathbf{P}^\pi\right)$ is a strictly diagonally dominant matrix (Syed et al., 2008). Thus, $\left(\mathbb{I} - \gamma \mathbf{P}^\pi\right)^{-1} = \sum_{t=0}^{\infty} (\gamma \mathbf{P}^\pi)^t$, where $\sum_{t=0}^{\infty} (\gamma \mathbf{P}^\pi)^t$ forms a valid *Neumann series* (Ward, 2021). We let $\mathbf{A}^\pi = \sum_{t=0}^{\infty} (\gamma \mathbf{P}^\pi)^t$, so Equation A.7 can be written as $\mathbf{u}_\pi = \mathbf{A}^\pi \mathbf{p}_0$. We can therefore express Equation A.4 in matrix form as:

$$\begin{aligned} \mathbf{v}_\pi &= \mathbf{\Pi} \odot \left(\mathbf{u}_\pi^T \otimes \mathbf{1}\right)^T \\ &= \mathbf{\Pi} \odot \left(\mathbf{p}_0^T (\mathbf{A}^\pi)^T \otimes \mathbf{1}\right)^T \,, \end{aligned} \tag{A.8}$$

where $\mathbf{\Pi}, \mathbf{v}_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$, $\mathbf{1} \in \mathbb{R}^{|\mathcal{A}|}$ is a column vector of ones, $\otimes$ presents the Kronecker product, and $\odot$ denotes the Hadamard product.

If we consider the case of a parameterized policy $\mathbf{\Pi}(\theta)$, then the derivative of $\mathbf{v}_\pi$ with respect to θ is

$$\begin{aligned} \nabla_\theta \mathbf{v}_\pi =& \nabla_\theta \left[\mathbf{\Pi} \odot \left(\mathbf{p}_0^T (\mathbf{A}^\pi)^T \otimes \mathbf{1}\right)^T\right] \\ =& \nabla_\theta \mathbf{\Pi} \odot \left(\mathbf{p}_0^T (\mathbf{A}^\pi)^T \otimes \mathbf{1}\right)^T + \mathbf{\Pi} \odot \nabla_\theta \left(\mathbf{p}_0^T (\mathbf{A}^\pi)^T \otimes \mathbf{1}\right)^T \\ =& \nabla_\theta \mathbf{\Pi} \odot \left(\mathbf{p}_0^T (\mathbf{A}^\pi)^T \otimes \mathbf{1}\right)^T + \mathbf{\Pi} \odot \left(\mathbf{p}_0^T (\nabla_\theta \mathbf{A}^\pi)^T \otimes \mathbf{1}\right)^T \end{aligned} \tag{A.9}$$

The first term in Equation A.9 is differentiable since the policy is parameterized by θ.

We expand $\nabla_\theta \mathbf{A}^\pi$ as follows:

$$\begin{aligned}
\nabla_\theta \mathbf{A}^\pi &= \nabla_\theta \left(\sum_{t=0}^{\infty} (\gamma \mathbf{P}^\pi)^t \right) \\
&= \nabla_\theta \left(\mathbb{I} + (\gamma \mathbf{P}^\pi) + (\gamma \mathbf{P}^\pi)^2 + \cdots + (\gamma \mathbf{P}^\pi)^t + \dots \right) \\
&= 0 + \nabla_\theta (\gamma \mathbf{P}^\pi) + 2(\gamma \mathbf{P}^\pi) \nabla_\theta (\gamma \mathbf{P}^\pi) + 3(\gamma \mathbf{P}^\pi)^2 \nabla_\theta (\gamma \mathbf{P}^\pi) + \cdots + t(\gamma \mathbf{P}^\pi)^{t-1} \nabla_\theta (\gamma \mathbf{P}^\pi) + \dots \\
&= \left(\mathbb{I} + 2(\gamma \mathbf{P}^\pi) + 3(\gamma \mathbf{P}^\pi)^2 + \cdots + t(\gamma \mathbf{P}^\pi)^{t-1} + \dots \right) \nabla_\theta (\gamma \mathbf{P}^\pi) \\
&= \left(\sum_{t=0}^{\infty} (t+1)(\gamma \mathbf{P}^\pi)^t \right) \nabla_\theta (\gamma \mathbf{P}^\pi) \\
&= \sum_{t=0}^{\infty} (t+1)(\gamma \mathbf{P}^\pi)^t \gamma \nabla_\theta \mathbf{P}^\pi \\
&\equiv \sum_{t=0}^{\infty} (t+1)(\gamma \mathbf{P}^\pi)^t \gamma \nabla_\theta \left[\sum_{s',a} T(s|s',a) \pi(a|s') \right] \\
&= \sum_{t=0}^{\infty} (t+1)(\gamma \mathbf{P}^\pi)^t \gamma \left[\sum_{s',a} T(s|s',a) \nabla_\theta \pi(a|s') \right].
\end{aligned} \tag{A.10}$$

Since Equation A.10 shows that $\nabla_\theta \mathbf{A}^\pi$ is differentiable, $\nabla_\theta \mathbf{v}_\pi$ is also differentiable based on Equation A.9. Proceeding similarly, given the same conditions, we see that all higher derivatives of $\mathbf{v}_\pi$ also exist with respect to θ. To show that $\mathcal{M}$ is a differentiable manifold, we note that the mapping $f : \Theta \to \mathcal{M}$ defined by $f(\theta) = v_{\pi_\theta}$ is a smooth map between the open parameter space $\Theta \subseteq \mathbb{R}^b$ and the space of probabilities distributions.

1. Smoothness - as derived above, $\mathbf{v}_\pi$ has derivatives of all orders with respect to θ, thus $f \in C^\infty$.

2. Homeomorphism - since the policy π_θ and occupancy measure v_{π_θ} have a bijective relationship (Syed et al., 2008; Givchi, 2021), then f is an embedding.

3. Immersion - provided the matrix $\nabla_\theta \mathbf{v}_\pi$ has a constant rank k for all $\theta \in \Theta$, f is a smooth immersion.

By the *Smooth Manifold Subspace Theorem* (Lee, 2009), if f is a smooth embedding, then its image $\mathcal{M} = f(\Theta)$ is a differentiable submanifold of $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$.

A.2 Proof of Proposition 3

In this section, for clarity of exposition, we use discrete states and actions to illustrate the proof. However, the formulation naturally extends to continuous state and action spaces — where summations are replaced by integrals — without additional assumptions (Puterman, 1994; Hernández-Lerma & Lasserre, 1996). Let us begin the proof by

defining the visitation probability at any step $h \in [H]$ in an episode, following policy $\pi(a|s)$. Specifically,

$$q_\pi^h(s,a) \triangleq \mathbb{P}(s_h = s, a_h = a) \;\; \forall h \in [H] \quad \text{and} \quad q_\pi^h(s,a) \triangleq 0 \;\; \forall h \in \mathbb{N} \wedge h > H\,. \tag{A.11}$$

Thus, we rewrite Equation 3.18 as $v_\pi^H(s,a) = \frac{1}{H}\sum_{h=1}^{H} q_\pi^h(s,a)$. Following (Kalagarla et al., 2021), we can write the Linear Programming formulation for solving episodic MDP $\mathbb{M}^H$ as

$$\begin{aligned}
&\max_{\{q_\pi^h\}_{h=1}^H} \sum_{h,s,a} r(s,a) q_\pi^h(s,a) \\
&\text{subject to} \sum_a q_\pi^h(s,a) = \sum_{s',a} T(s \mid s',a) q_\pi^{h-1}(s',a) \quad \forall h \in [H] \wedge h > 1\,, \\
&\qquad\qquad q_\pi^1(s,a) = \pi(a|s)\mu(s)\,, \\
&\qquad\qquad q_\pi^h(s,a) \geq 0 \qquad\qquad \forall h \in [H], (s,a) \in \mathcal{S} \times \mathcal{A}\,,
\end{aligned} \tag{A.12}$$

where $\mu(s)$ is the initial state distribution and $T(s \mid s', a)$ is the transition probability. The constraints of this optimization problem are often referred to as *Bellman Flow Constraints*.

This implies that

$$\begin{aligned}
&\sum_{h=2}^{H+1}\sum_a q_\pi^h(s,a) = \sum_{h=2}^{H+1}\sum_{s',a} T(s \mid s',a) q_\pi^{h-1}(s',a) \\
\implies &\sum_a q_\pi^1(s,a) + \sum_{h=2}^{H+1}\sum_a q_\pi^h(s,a) = \sum_{h=2}^{H+1}\sum_{s',a} T(s \mid s',a) q_\pi^{h-1}(s',a) + \sum_a q_\pi^1(s,a) \\
\implies &\sum_a \sum_{h=1}^{H+1} q_\pi^h(s,a) = \sum_{h=2}^{H+1}\sum_{s',a} T(s \mid s',a) q_\pi^{h-1}(s',a) + \sum_a q_\pi^1(s,a) \\
\implies &H\sum_a v_\pi^H(s,a) = \sum_{s',a} T(s \mid s',a)(\sum_{h=2}^{H+1} q_\pi^{h-1}(s',a)) + \mu(s) \\
\implies &H\sum_a v_\pi^H(s,a) = H\sum_{s',a} T(s \mid s',a) v_\pi^H(s',a) + \mu(s) \\
\implies &\sum_a v_\pi^H(s,a) = \sum_{s',a} T(s \mid s',a) v_\pi^H(s',a) + \frac{1}{H}\mu(s) \\
\implies &u_\pi^H(s) \triangleq \sum_a v_\pi^H(s,a) = \sum_{s',a} T(s \mid s',a)\pi(a|s') u_\pi^H(s') + \frac{1}{H}\mu(s)\,.
\end{aligned} \tag{A.13}$$

Now, we denote $\mathbf{u}_\pi^H$ and $\bar{\mu}$ as corresponding column vectors and the transition matrix $\mathbf{P}^\pi \triangleq \left[\sum_{s',a} T(s \mid s',a)\pi(a|s')\right]$. Thus, we obtain

$$(\mathbb{I} - \mathbf{P}^\pi)\mathbf{u}_\pi^H = \frac{1}{H}\bar{\mu}. \tag{A.14}$$

We can therefore express the finite horizon occupancy measure in matrix form as

$$\mathbf{v}_\pi^H = \mathbf{\Pi} \odot \left((\mathbf{u}_\pi^H)^T \otimes \mathbf{1}\right)^T \tag{A.15}$$

where $\mathbf{\Pi}, \mathbf{v}_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$, $\mathbf{1} \in \mathbb{R}^{|\mathcal{A}|}$ is a column vector of ones, $\otimes$ presents the Kronecker product, $\odot$ denotes the Hadamard product.

If we consider the case of a parameterized policy $\mathbf{\Pi}(\theta)$, the derivative of $\mathbf{v}_\pi^H$ with respect to θ is

$$\begin{aligned}\nabla_\theta \mathbf{v}_\pi^H =& \nabla_\theta \left[\mathbf{\Pi} \odot \left((\mathbf{u}_\pi^H)^T \otimes \mathbf{1}\right)^T\right] \\ =& \nabla_\theta \mathbf{\Pi} \odot \left((\mathbf{u}_\pi^H)^T \otimes \mathbf{1}\right)^T + \mathbf{\Pi} \odot \nabla_\theta \left((\mathbf{u}_\pi^H)^T \otimes \mathbf{1}\right)^T \\ =& \nabla_\theta \mathbf{\Pi} \odot \left((\mathbf{u}_\pi^H)^T \otimes \mathbf{1}\right)^T + \mathbf{\Pi} \odot \left((\nabla_\theta \mathbf{u}_\pi^H)^T \otimes \mathbf{1}\right)^T\end{aligned} \tag{A.16}$$

The first term in Equation A.16 is differentiable since the policy is parameterized by θ. We show that $\nabla_\theta \mathbf{u}_\pi^H$ exists using Equation A.14 as follows:

$$\begin{aligned}\nabla_\theta \left[(\mathbb{I} - \mathbf{P}^\pi)\mathbf{u}_\pi^H\right] &= \frac{1}{H}\nabla_\theta \bar{\mu} \\ \nabla_\theta(\mathbb{I} - \mathbf{P}^\pi)\mathbf{u}_\pi^H + (\mathbb{I} - \mathbf{P}^\pi)\nabla_\theta \mathbf{u}_\pi^H &= 0 \\ -\nabla_\theta \mathbf{P}^\pi \mathbf{u}_\pi^H + (\mathbb{I} - \mathbf{P}^\pi)\nabla_\theta \mathbf{u}_\pi^H &= 0 \\ (\mathbb{I} - \mathbf{P}^\pi)\nabla_\theta \mathbf{u}_\pi^H &= \nabla_\theta \mathbf{P}^\pi \mathbf{u}_\pi^H\end{aligned} \tag{A.17}$$

In Equation A.17, we observe that $(\mathbb{I} - \mathbf{P}^\pi)^{-1}$ may or may not exist. To address this, we use the group inverse, a generalized matrix inverse that extends the concept of inversion to both singular and invertible matrices (Drazin, 1958). For a square matrix $\mathbf{C} \in \mathbb{R}^{n\times n}$, its group inverse $\mathbf{C}^\#$ satisfies the conditions $\mathbf{C}\mathbf{C}^\#\mathbf{C} = \mathbf{C}$, $\mathbf{C}^\#\mathbf{C}\mathbf{C}^\# = \mathbf{C}^\#$ and $\mathbf{C}\mathbf{C}^\# = \mathbf{C}^\#\mathbf{C}$.

The group inverse is useful in analysing Markov chains (Meyer, 1975) and coincides with the standard inverse when $\mathbf{C}$ is invertible. Using group inverse $(\mathbb{I}-\mathbf{P}^\pi)^\#$, Equation A.17 can be expressed as,

$$\nabla_\theta \mathbf{u}_\pi^H = (\mathbb{I} - \mathbf{P}^\pi)^\# \nabla_\theta \mathbf{P}^\pi \mathbf{u}_\pi^H + \mathbf{w} \tag{A.18}$$

where $\mathbf{w}$ is a vector in the null space of $(\mathbb{I} - \mathbf{P}^\pi)$. Note that $\mathbf{w} = \mathbf{0}$ for ergodic Markov chains (Meyer, 1975). An episodic MDP induces an ergodic Markov chain that admits a unique stationary distribution (Bojun, 2020) (also see Section 3.5.3). Thus, Equation A.18 simplifies to

$$\nabla_\theta \mathbf{u}_\pi^H = (\mathbb{I} - \mathbf{P}^\pi)^\# \nabla_\theta \mathbf{P}^\pi \mathbf{u}_\pi^H \tag{A.19}$$

$$= (\mathbb{I} - \mathbf{P}^\pi)^\# \left[\sum_{s',a} T(s|s',a)\nabla_\theta \pi(a|s')\right] \mathbf{u}_\pi^H \tag{A.20}$$

which shows that $\nabla_\theta \mathbf{u}_\pi^H$ is differentiable and so is $\nabla_\theta \mathbf{v}_\pi^H$. Proceeding similarly, given the same conditions, we see that all higher derivatives of $\mathbf{v}_\pi^H$ also exist with respect to θ.

To show that $\mathcal{M}^H$ is a differentiable manifold, we note that the mapping $f : \Theta \to \mathcal{M}^H$ defined by $f(\theta) = v_\pi^H$ is a smooth map between the open parameter space $\Theta \subseteq \mathbb{R}^b$ and the space of probability measures.

1. Smoothness - as derived above, $\mathbf{v}_\pi^H$ has derivatives of all orders with respect to θ, therefore $f \in C^\infty$.

2. Homeomorphism - since the policy π_θ and occupancy measure v_π have a bijective relationship, f is an embedding.

3. Immersion - provided the matrix $\nabla_\theta \mathbf{v}_\pi^H$ has constant rank k for all $\theta \in \Theta$, then f is a smooth immersion.

By the *Smooth Manifold Subspace Theorem* (Lee, 2009), if f is a smooth embedding, then its image $\mathcal{M}^H = f(\Theta)$ is a differentiable submanifold of $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$.

A.3 Proof of Proposition 4

We compute the error in occupancy measure for both the infinite and finite horizon cases. In infinite horizon MDPs, the occupancy measure is defined as the expected discounted number of visits of a state-action pair (s, a) in a trajectory (Laroche & des Combes, 2023): $\mu = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \mu_t$, where $\mu_t = P(s_t, a_t \mid \pi, \eta)$ is the state-action probability distribution at time step t with the initial state distribution η following the policy π. In finite horizon MDPs, the occupancy measure is the expected number of visits of a state-action pair (s, a) in an episode of length H (Altman, 1999): $\mu = \frac{1}{H}\sum_{t=1}^{H} \mu_t$.

First, we derive error bounds for the infinite horizon MDP in which $\gamma < 1$ and the occupancy measure is approximated using a finite number of samples collected up to a finite number of time steps T. Later, we derive error bounds for the finite horizon MDP.

A.3.1 Infinite Horizon MDPs

Estimated Occupancy Measure. To compute μ, we roll out N episodes (each of multiple time steps) using π, and take N number of samples at t to approximate μ_t. Thus, the empirical occupancy measure $\hat{\mu}$ is given by $\hat{\mu} = \rho \sum_{t=0}^{T} \gamma^t \hat{\mu}_t^N$, where $\rho = \frac{1}{\sum_{t=0}^{T} \gamma^t}$. Note that the total number of samples in the policy dataset $\mathcal{D}_\pi$ is $|\mathcal{D}_\pi| = N(T+1)$.

Occupancy Measure Estimation Error. Consider two occupancy measures $\mu = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \mu_t$ and $\nu = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \nu_t$ (with estimates $\hat{\mu} = \rho \sum_{t=0}^{T} \gamma^t \hat{\mu}_t^{N_\mu}$ and $\hat{\nu} = \rho \sum_{t=0}^{T} \gamma^t \hat{\nu}_t^{N_\nu}$). For independent sets $\{\mu_t\}_{t\geq 0}$ and $\{\nu_t\}_{t\geq 0}$, the Wasserstein distance

has the following additive property (Panaretos & Zemel, 2019),

$$\mathcal{W}_p(\sum_t \mu_t, \sum_t \nu_t) \leq \sum_t \mathcal{W}_p(\mu_t, \nu_t) \tag{A.21}$$

While for $a \in \mathbb{R}$ (Panaretos & Zemel, 2019),

$$\mathcal{W}_p(a\mu, av) = |a|\mathcal{W}_p(\mu, v) \tag{A.22}$$

Therefore, for our scenario where $p = 1$, the Wasserstein distance between μ and ν is given by:

$$\begin{aligned} \mathcal{W}_1(\mu, \nu) &= \mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\mu_t, (1-\gamma)\sum_{t=0}^{\infty}\gamma^t\nu_t) \\ &\leq (1-\gamma)\sum_{t=0}^{\infty}\gamma^t\mathcal{W}_1(\mu_t, \nu_t) \end{aligned} \tag{A.23}$$

while for $\hat{\mu}$ and $\hat{\nu}$,

$$\mathcal{W}_1(\hat{\mu}, \hat{\nu}) \leq \rho\sum_{t=0}^{T}\gamma^t\mathcal{W}_1(\hat{\mu}_t^{N_\mu}, \hat{\nu}_t^{N_\nu}) \tag{A.24}$$

In the RL problems we consider, the state-action space $\mathcal{Z} = \mathcal{S} \times \mathcal{A}$ is commonly defined as the subset of the Euclidean space $\mathcal{Z} \in \mathbb{R}^B$, where usually $B \geq 2$. Theorems 1 and 3 in (Sommerfeld et al., 2019) establish the following error bounds between the true and empirical probability distributions,

$$\begin{aligned} \mathbb{E}[\mathcal{W}_1(\hat{\mu}_t^{N_\mu}, \mu_t)] &\leq \mathcal{E}_2 N_\mu^{-\frac{1}{2}} \\ \mathbb{E}[\mathcal{W}_1(\hat{\nu}_t^{N_\nu}, \nu_t)] &\leq \mathcal{E}_2 N_\nu^{-\frac{1}{2}} \end{aligned} \tag{A.25}$$

where

$$\mathcal{E}_2 \leq 4B^{1/2} diam(\mathcal{Z}) \cdot \begin{cases} 2 + (1/2)\log_2|\mathcal{Z}| & \text{if } B = 2 \\ |\mathcal{Z}|^{1/2-1/B}\left[2 + 1/(2^{B/2-1}-1)\right] & \text{if } B > 2 \end{cases}$$

Note that $|\mathcal{Z}|$ and $diam(\mathcal{Z})$ denote the cardinality and diameter of $\mathcal{Z}$, respectively.

Suppose $a = \mathcal{W}_1(\hat{\mu}, \hat{\nu})$, $b = \mathcal{W}_1(\hat{\mu}, \mu)$, $c = \mathcal{W}_1(\hat{\nu}, \mu)$, $d = \mathcal{W}_1(\mu, \nu)$, and $e = \mathcal{W}_1(\hat{\nu}, \nu)$. Then by performing two reverse triangle inequalities,

$$\begin{aligned} &|a - c| \leq b \quad \text{and} \quad |c - d| \leq e \\ \implies &|a - d| \leq b + e \end{aligned} \tag{A.26}$$

Equation A.26 implies that,

$$\begin{aligned}\mathbb{E}[|\mathcal{W}_1(\hat{\mu},\hat{\nu})-\mathcal{W}_1(\mu,\nu)|] &\leq \mathbb{E}[\mathcal{W}_1(\hat{\mu},\mu)+\mathcal{W}_1(\hat{\nu},\nu)] \\ &= \mathbb{E}[\mathcal{W}_1(\rho\sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu},\mu)+\mathcal{W}_1(\rho\sum_{t=0}^{T}\gamma^t\hat{\nu}_t^{N_\nu},\nu)] \\ &= \mathbb{E}[\mathcal{W}_1(\rho\sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu},\mu)]+\mathbb{E}[\mathcal{W}_1(\rho\sum_{t=0}^{T}\gamma^t\hat{\nu}_t^{N_\nu},\nu)] \\ &+ \mathbb{E}[\mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu},\mu)-\mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu},\mu)] \\ &+ \mathbb{E}[\mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\nu}_t^{N_\nu},\nu)-\mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\nu}_t^{N_\nu},\nu)]\end{aligned} \tag{A.27}$$

By virtue of triangle inequalities, we get

$$\begin{aligned}\mathcal{W}_1(\rho\sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu},(1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu}) &\geq \mathcal{W}_1(\rho\sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu},\mu)-\mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu},\mu) \\ \mathcal{W}_1(\rho\sum_{t=0}^{T}\gamma^t\hat{\nu}_t^{N_\nu},(1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\nu}_t^{N_\nu}) &\geq \mathcal{W}_1(\rho\sum_{t=0}^{T}\gamma^t\hat{\nu}_t^{N_\nu},\nu)-\mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\nu}_t^{N_\nu},\nu)\end{aligned} \tag{A.28}$$

Therefore, the right-hand-side (R.H.S) of Equation A.27 can be further simplified as

$$\begin{aligned}\text{R.H.S} &\leq \mathbb{E}[\mathcal{W}_1(\rho\sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu},(1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu})]+\mathbb{E}[\mathcal{W}_1(\rho\sum_{t=0}^{T}\gamma^t\hat{\nu}_t^{N_\nu},(1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\nu}_t^{N_\nu})] \\ &+ \mathbb{E}[\mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu},\mu)]+\mathbb{E}[\mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\nu}_t^{N_\nu},\nu)]\end{aligned} \tag{A.29}$$

For simplicity, we denote $\hat{\mu}_\infty = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu}$ (similarly $\hat{\nu}_\infty$) and $\hat{\mu}_T = \rho\sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu}$ (similarly $\hat{\nu}_T$), where $\rho = \frac{1}{\sum_{t=0}^{T}\gamma^t} = \frac{1-\gamma}{1-\gamma^{T+1}}$. Using Theorem 4 in (Gibbs & Su, 2002), the 1-Wasserstein metric $\mathcal{W}_1$ and the total variation distance d_{TV} satisfy the following,

$$\begin{aligned}\mathcal{W}_1(\hat{\mu}_\infty,\hat{\mu}_T) &\leq diam(\mathcal{Z})\cdot d_{TV}(\hat{\mu}_\infty,\hat{\mu}_T) \\ &= diam(\mathcal{Z})\cdot\frac{1}{2}\sum_{z\in\mathcal{Z}}|\hat{\mu}_\infty(z)-\hat{\mu}_T(z)|\end{aligned} \tag{A.30}$$

However,

$$\begin{aligned}
\hat{\mu}_\infty - \hat{\mu}_T &= (1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu} - \frac{1-\gamma}{1-\gamma^{T+1}}\sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu} \\
&= (1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu} - \frac{1-\gamma}{1-\gamma^{T+1}}\sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu} \\
&+ (1-\gamma)\sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu} - (1-\gamma)\sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu} \\
&= (1-\gamma)\left(\sum_{t=0}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu} - \sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu}\right) + \left((1-\gamma) - \frac{1-\gamma}{1-\gamma^{T+1}}\right)\sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu} \\
&= (1-\gamma)\sum_{t=T+1}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu} - \gamma^{T+1}\frac{1-\gamma}{1-\gamma^{T+1}}\sum_{t=0}^{T}\gamma^t\hat{\mu}_t^{N_\mu} \\
&\leq (1-\gamma)\sum_{t=T+1}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu} \\
&= \gamma^{T+1}\frac{1-\gamma}{\gamma^{T+1}}\sum_{t=T+1}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu} \\
&= \gamma^{T+1}\hat{\mu}_{T+1,\infty}
\end{aligned} \tag{A.31}$$

where $\frac{1-\gamma}{\gamma^{T+1}}$ normalizes $\sum_{t=T+1}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu}$. We utilize Equation A.31 in Equation A.30 as,

$$\begin{aligned}
\mathcal{W}_1(\hat{\mu}_\infty, \hat{\mu}_T) &\leq diam(\mathcal{Z}) \cdot \frac{1}{2}\sum_{z\in\mathcal{Z}}|\hat{\mu}_\infty(z) - \hat{\mu}_T(z)| \\
&\leq diam(\mathcal{Z}) \cdot \frac{1}{2}\sum_{z\in\mathcal{Z}}|\gamma^{T+1}\hat{\mu}_{T+1,\infty}(z)| \\
&= \frac{\gamma^{T+1}}{2}diam(\mathcal{Z})
\end{aligned} \tag{A.32}$$

Equation A.32 also applies for $\mathcal{W}_1(\hat{\nu}_\infty, \hat{\nu}_T)$, therefore by substituting these into Equa-

tion A.29,

$$\begin{aligned}
\text{R.H.S} &\leq \mathbb{E}[\mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu},\mu)] + \mathbb{E}[\mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\nu}_t^{N_\nu},\nu)] + \gamma^{T+1}diam(\mathcal{Z}) \\
&= \mathbb{E}[\mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\mu}_t^{N_\mu},(1-\gamma)\sum_{t=0}^{\infty}\gamma^t\mu_t)] \\
&+ \mathbb{E}[\mathcal{W}_1((1-\gamma)\sum_{t=0}^{\infty}\gamma^t\hat{\nu}_t^{N_\nu},(1-\gamma)\sum_{t=0}^{\infty}\gamma^t\nu_t)] + \gamma^{T+1}diam(\mathcal{Z}) \\
&\leq (1-\gamma)\sum_{t=0}^{\infty}\gamma^t\left(\mathbb{E}[\mathcal{W}_1(\hat{\mu}_t^{N_\mu},\mu_t)] + \mathbb{E}[\mathcal{W}_1(\hat{\nu}_t^{N_\mu},\nu_t)]\right) + \gamma^{T+1}diam(\mathcal{Z})\,.
\end{aligned} \tag{A.33}$$

By substituting Equation A.25 into Equation A.33

$$\begin{aligned}
\text{R.H.S} &\leq (1-\gamma)\sum_{t=0}^{\infty}\gamma^t\left(\mathcal{E}_2 N_\mu^{-\frac{1}{2}} + \mathcal{E}_2 N_\nu^{-\frac{1}{2}}\right) + \gamma^{T+1}diam(\mathcal{Z}) \\
&= \mathcal{E}_2\left(N_\mu^{-\frac{1}{2}} + N_\nu^{-\frac{1}{2}}\right) + \gamma^{T+1}diam(\mathcal{Z})
\end{aligned} \tag{A.34}$$

Therefore, Equation A.27 becomes:

$$\mathbb{E}[|\mathcal{W}_1(\hat{\mu},\hat{\nu}) - \mathcal{W}_1(\mu,\nu)|] \leq \mathcal{E}_2\left(N_\mu^{-\frac{1}{2}} + N_\nu^{-\frac{1}{2}}\right) + \gamma^{T+1}diam(\mathcal{Z}) \tag{A.35}$$

Over the full trajectory in the occupancy measure space. The true distance between consecutive policies π_i and π_{i+1} after an update is $\mathcal{W}_1(v_{\pi_i}, v_{\pi_{i+1}})$, which is induced by the i^{th} policy update. We estimate this distance using datasets of the policies, i.e. approximated distributions, using $\mathcal{W}_1(\hat{v}_{\pi_i}, \hat{v}_{\pi_{i+1}})$.

For M roll out episodes of each π_i, we use Equation A.35, with $N_\mu = N_\nu = M$, to derive the following error bounds,

$$\mathbb{E}\left[\left|\mathcal{W}_1(v_{\pi_i}, v_{\pi_{i+1}}) - \mathcal{W}_1(\hat{v}_{\pi_i}, \hat{v}_{\pi_{i+1}})\right|\right] \leq 2\mathcal{E}_2 M^{-\frac{1}{2}} + \gamma^{T+1}diam(\mathcal{Z}) \tag{A.36}$$

which is consistent with learning from $\mathcal{D}_{\pi_i}$ and then $\mathcal{D}_{\pi_{i+1}}$. By summing sequentially through policies encountered during RL training, we compute the total distance over a path of N segments obtained via policy updates:

$$\sum_{i=0}^{N-1}\mathbb{E}\left[\left|\mathcal{W}_1(v_{\pi_i}, v_{\pi_{i+1}}) - \mathcal{W}_1(\hat{v}_{\pi_i}, \hat{v}_{\pi_{i+1}})\right|\right] \leq 2N\mathcal{E}_2 M^{-\frac{1}{2}} + N\gamma^{T+1}diam(\mathcal{Z}) \tag{A.37}$$

Since $|\sum_t x_t| \leq \sum_t |x_t|$ then,

$$\mathbb{E}\left[\left|\sum_{i=0}^{N-1}\mathcal{W}_1(v_{\pi_i}, v_{\pi_{i+1}}) - \sum_{i=0}^{N-1}\mathcal{W}_1(\hat{v}_{\pi_i}, \hat{v}_{\pi_{i+1}})\right|\right] \leq \frac{2N\mathcal{E}_2}{\sqrt{M}} + N\gamma^{T+1}diam(\mathcal{Z}) \tag{A.38}$$

A.3.2 Finite Horizon MDPs

Occupancy Measure Estimated Error. Consider two occupancy measures $\mu = \frac{1}{H}\sum_{t=1}^{H}\mu_t$ and $\nu = \frac{1}{H}\sum_{t=1}^{H}\nu_t$ with estimates $\hat{\mu} = \frac{1}{H}\sum_{t=1}^{H}\hat{\mu}_t^{N_\mu}$ and $\hat{\nu} = \frac{1}{H}\sum_{t=1}^{H}\hat{\nu}_t^{N_\nu}$. From Equation A.26, we have

$$\begin{aligned}
&\mathbb{E}[|\mathcal{W}_1(\hat{\mu},\hat{\nu}) - \mathcal{W}_1(\mu,\nu)|] \\
&\leq \mathbb{E}[\mathcal{W}_1(\hat{\mu},\mu) + \mathcal{W}_1(\hat{\nu},\nu)] \\
&= \mathbb{E}[\mathcal{W}_1(\frac{1}{H}\sum_{t=1}^{H}\hat{\mu}_t^{N_\mu}, \frac{1}{H}\sum_{t=1}^{H}\mu_t) + \mathcal{W}_1(\frac{1}{H}\sum_{t=1}^{H}\hat{\nu}_t^{N_\nu}, \frac{1}{H}\sum_{t=1}^{H}\nu_t)] \\
&\leq \frac{1}{H}\sum_{t=1}^{H}\mathbb{E}[\mathcal{W}_1(\hat{\mu}_t^{N_\mu},\mu_t)] + \frac{1}{H}\sum_{t=1}^{H}\mathbb{E}[\mathcal{W}_1(\hat{\nu}_t^{N_\nu},\nu_t)] \\
&\leq \mathcal{E}_2\left(N_\mu^{-\frac{1}{2}} + N_\nu^{-\frac{1}{2}}\right)
\end{aligned} \tag{A.39}$$

Therefore **for the total path in the occupancy measure space** with M roll out episodes of each π_i, the error bound is

$$\mathbb{E}\left[\left|\sum_{i=0}^{N-1}\mathcal{W}_1(v_{\pi_i}, v_{\pi_{i+1}}) - \sum_{i=0}^{N-1}\mathcal{W}_1(\hat{v}_{\pi_i}, \hat{v}_{\pi_{i+1}})\right|\right] \leq \frac{2N\mathcal{E}_2}{\sqrt{M}} \tag{A.40}$$

by assigning $N_\mu = N_\nu = M$ in Equation A.39, which concludes the proof. Note that the proofs in this section naturally extend to continuous state and action spaces (where summations are replaced by integrals) without making additional assumptions.

Appendix B

Empirical Analysis of Exploration

This section contains additional empirical results of Chapter 3.

B.1 η_{sub} can be a reasonable proxy for η, when optimal policy is not fully reached

We compare ESL when the optimal policy was reached, denoted η, versus when it was not, denoted η_{sub}, in Tables B.1 and B.2. First, we observe that the number of rollouts impacts the metric values. Second, η_{sub} values are always greater than η values. Note that UCRL2 and PSRL update their policies only at the end of each episode, whereas SAC and DQN update theirs after each time step. Hence, $\text{UC}_{sub} = 499$ for both UCRL2 and PSRL.

The ESL values (both η and η_{sub}) in Table B.2 are lower than those in Table B.1, as expected since more data samples reduce estimation error. The distance from the initial policies to the final polices are not so different. Using both Tables B.1 and B.2, we notice that comparing algorithms with η_{sub} yields the same efficiency ranking (e.g. PSRL, UCRL, SAC and DQN) as η. This indicates that η_{sub} reliably predicts results provided by η for comparing algorithms.

The results presented in Table 3.2 for stochastic dense-rewards setting are consistent with those in Table B.2 because the number of rollouts used was $\text{Nr} = 6$.

Algo.	η	η_{sub}	d	c	UC	UC_{sub}
SAC	849± 468	3623± 4166	5.63± 1.50	5.26± 2.10	1850± 742	7451± 3535
UCRL2	230± 155	613± 999	5.65± 0.93	5.45± 2.15	284± 180	499± 0.0
PSRL	86.2± 44.4	389± 102	4.96± 1.26	5.29± 1.49	97.2± 52.5	499± 0.0
DQN	564± 478	3911± 1710	5.52± 1.39	6.54± 2.05	1213± 1061	9097± 1904

Table B.1: *Evaluation of algorithms in the* ***stochastic, dense-rewards setting*** *for* 5×5 *gridworld with* 40 ***maximum steps per episode*** *with the number of rollouts* $Nr = 1$*. The total number of training episodes is* 500*. When the algorithm converged at optimality,* η *is the Effort of Sequential Learning,* $d = \mathcal{W}_1(\pi_0, \pi^*)$ *is distance from initial policy to the optimal policy, and UC is the number of updates to convergence. When the algorithm did not converge at the optimal policy, rather a non-optimal* π_N*, we use* η_{sub}*,* $c = \mathcal{W}_1(\pi_0, \pi_N)$*, and* UC_{sub} *to denote the aforementioned quantities.* 40 *training trials were used.*

Algo.	η	η_{sub}	d	c	UC	UC_{sub}
SAC	445± 246	853± 127	5.63± 1.23	7.26± 1.45	2463± 2043	6293± 441
UCRL2	198± 121	510± 274	5.36± 0.84	4.58± 1.90	268± 155	499± 0.0
PSRL	55.4± 33.6	361± 43.6	4.97± 1.34	3.91± 0.48	76.1± 50.6	499± 0.0
DQN	458± 311	1971± 250	4.88± 1.06	6.52± 0.31	1586± 1077	13713± 6907

Table B.2: *Evaluation of algorithms in the* ***stochastic, dense-rewards setting*** *for* 5×5 *gridworld with* 40 ***maximum steps per episode*** *with the number of rollouts* $Nr = 6$*. The total number of training episode is* 500*. When the algorithm converged at optimality,* η *is the Effort of Sequential Learning,* $d = \mathcal{W}_1(\pi_0, \pi^*)$ *is distance from initial policy to the optimal policy, and UC is the number of updates to convergence. When the algorithm did not converge at the optimal policy however some* π_N*, we use* η_{sub}*,* $c = \mathcal{W}_1(\pi_0, \pi_N)$*, and* UC_{sub} *to denote the aforementioned quantities.* 40 *training trials were used.*

B.2 Computing Occupancy Measure in Episodic Settings

Figure B.1 depicts *Rel. Error%* vs *number of updates* in a simple 5×5 2D-Gridworld environment. We observe that increasing the number of rollouts M reduces the estimation error of v_π^H. For $M = 10$, the absolute relative error can be as high as 50% with the mean less than 10%. While for $M = 500$, the maximum absolute relative error is 4%.

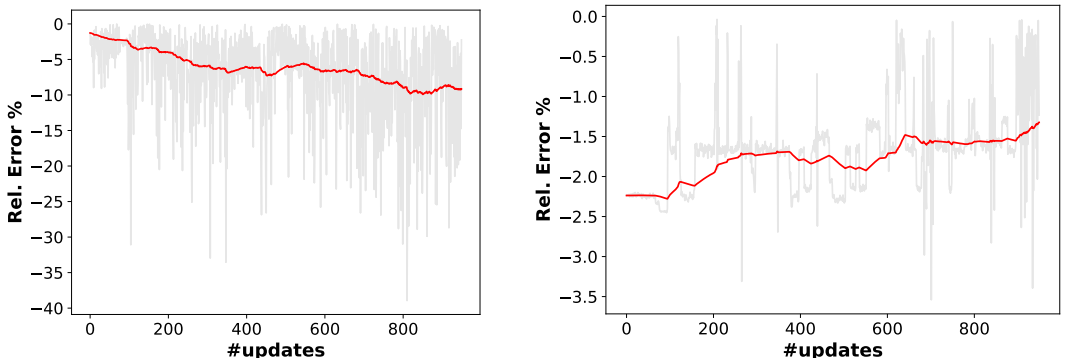


Figure B.1: *Rel. Error% vs number of updates plots in a 5×5 2D-Gridworld environment with noisy actions and dense rewards, where v_π^H is estimated using $M = 10$ rollouts (left) and $M = 500$ rollouts (right).*

B.3 Limitations of determining the mapping $u \to \alpha_u(T)$

In practice, mapping labels u to distributions over the feature space $\mathcal{P}(\mathcal{T})$ as $u \to \alpha_u(T) = P(T \mid U = u) \in \mathcal{P}(\mathcal{T})$ presents some challenges. These include (1) *support limitation*, (2) *curse of dimensionality*, and (3) *non-surjectivity of the mapping*. In *support limitation*, we observe that if some u occurs rarely, then $\alpha_u(T)$ is poorly estimated. In *curse of dimensionality*, we note that if $U \in \mathbb{R}^d$ and d is large, then large amounts of data samples are required to estimate $\alpha_u(T)$. That is, the sample complexity of estimating $\alpha_u(T)$ scales with the dimension of u.

In *non-surjectivity of the mapping*, we observe that the mapping $u \to \alpha_u(T)$ is generally not surjective onto the full space of probability distributions over T. Rather the mapping is restricted to densities derivable from the underlying joint distribution that satisfies $P(U = u, T = t) = P(U = u) \cdot \alpha_u(T)$. Thus, we assume that $\{\alpha_u(T)\}_{u \in U}$ occupies a subset of the distribution space that any density estimator can approximate on the support of the data. In discrete settings (where u is discrete), we assume that the true $\alpha_u(T)$ is identifiable from finite counts on the observed support. In continuous settings (where u is continuous), we assume that the true $\alpha_u(T)$ varies smoothly with u such that the neighbourhoods of u produce conditional distributions over T that are close in terms of the Wasserstein distance.

B.4 Supplementary Results

In this section we present enlarged versions of results in Figure 3.5 (see Section B.4.1) and additional plots that support the results in the main paper (see Section B.4.3).

B.4.1 Enlarged Visualisation of the Occupancy Measure Trajectories

Figures B.2 - B.4 are enlarged versions of enlarged versions of Figure 3.5. For each algorithm, there is a visualisation of the policy trajectory and visualisation of the state visitation below it.

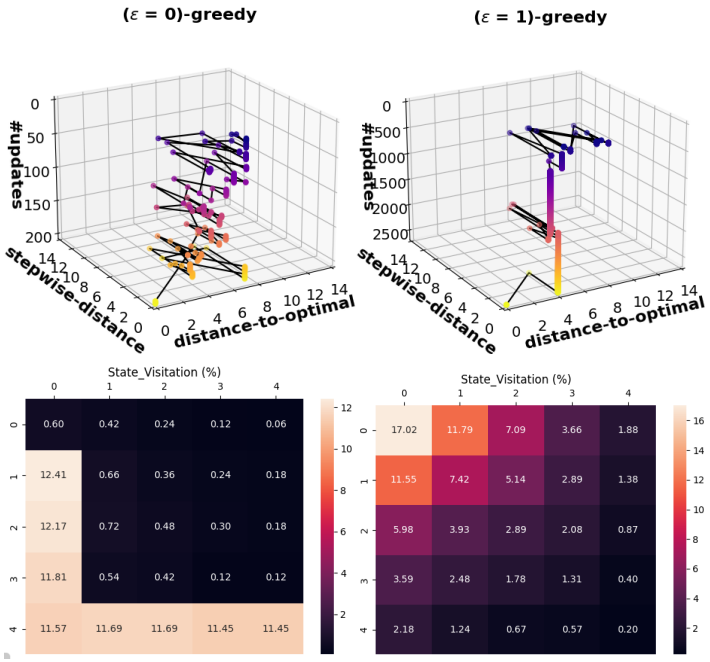


Figure B.2: *Top row: Scatter plots of distance-to-optimal and stepwise-distance over updates for $\epsilon(=0)$-greedy and $\epsilon(=1)$-greedy Q-learning. Bottom row: State visitations.*

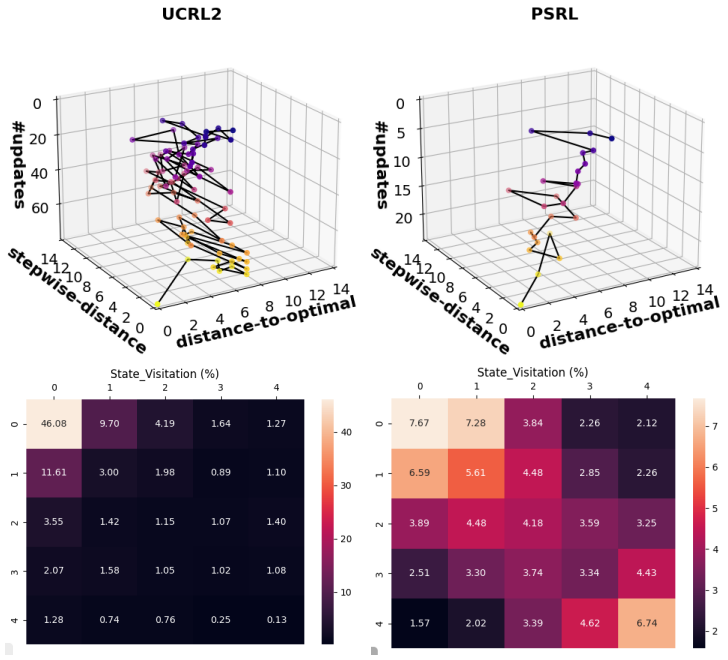


Figure B.3: *Top row: Scatter plots of distance-to-optimal and stepwise-distance over updates for UCRL2 and PSRL. Bottom row: State visitations.*

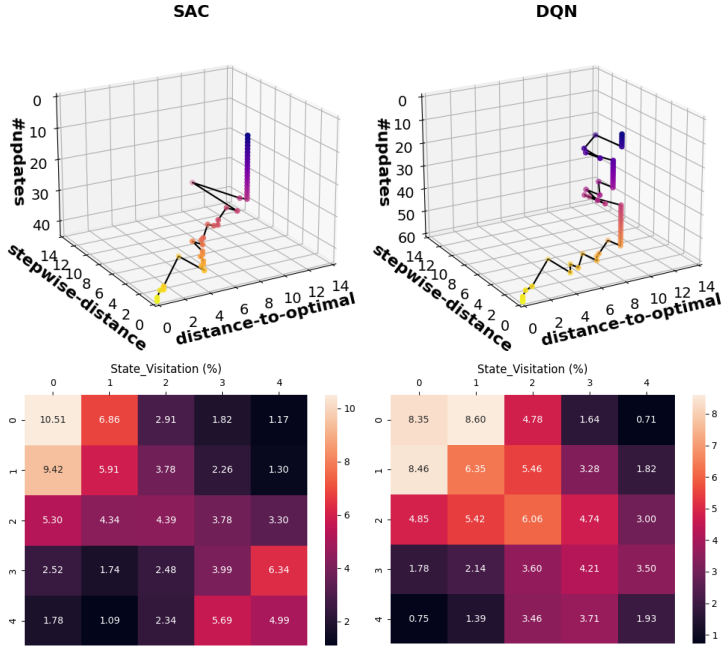
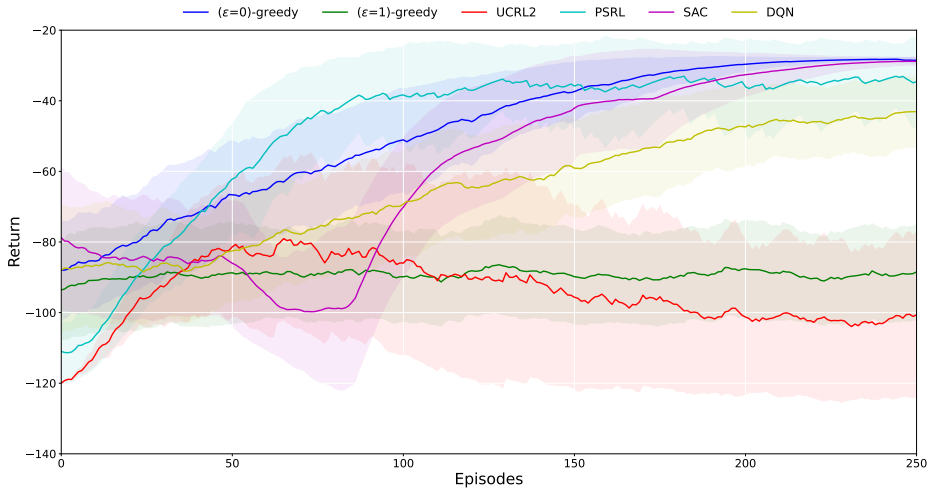


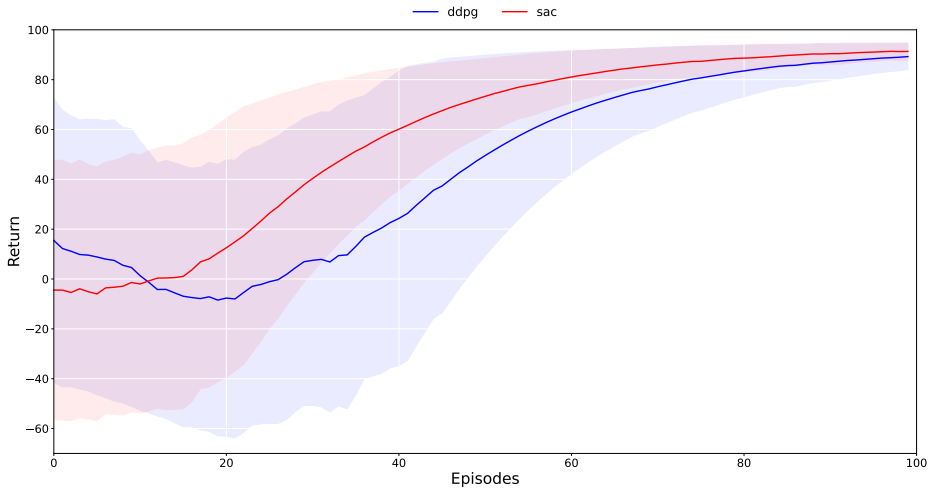
Figure B.4: *Top row: Scatter plots of distance-to-optimal and stepwise-distance over updates for SAC and DQN. Bottom row: State visitations.*

B.4.2 Performance Plots

This section contains Return plots of the algorithms. This allows us to assess the learning of algorithms from the performance perspective. Figures B.5a and B.5b depict performance evolution that corresponds to settings in Figures 3.5 and 3.6, respectively. Note that while all algorithms find the optimal policy, UCRL2 and $\epsilon(=1)$-greedy Q-learning fail to remain there if training continues without truncation. As a result, their performance does not improve over time compared to those that stabilize at the optimal policy.



(a)



(b)

Figure B.5: *(a) Return plots of algorithms: $\epsilon(=0)$-greedy and $\epsilon(=1)$-greedy Q-learning, UCRL2, PSRL, SAC, and DQN averaged over 5 runs in the deterministic 5×5 Gridworld with dense rewards. (b) Return plots of algorithms: DDPG and SAC averaged over 5 runs in the continuous Mountain Car problem.*

B.4.3 Evolution of *stepwise-distance*, *distance-to-optimal*, and OMR(k)

In this section we present 2 dimensional versions of the policy trajectories in Figures 3.5 and 3.6, along with corresponding OMR evolution plots. These are *stepwise-distance* vs. updates, *distance-to-optimal* vs. updates, and OMR(k) plots for the algorithms. Figure B.6 presents plots for the continuous environment Mountain Car, while Figure B.7) presents plots for the discrete environment 2D Gridworld.

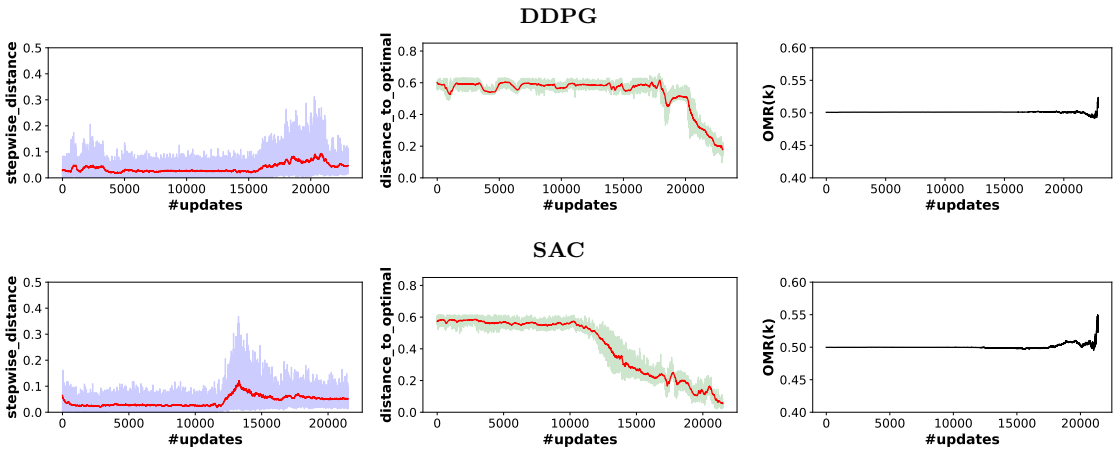


Figure B.6: *Plots in the first column are stepwise-distance vs. number of updates, second column distance-to-optimal vs. number of updates, and third OMR(k) vs. number of updates. Top row plots belong to DDPG algorithm, while bottom row plots belong to SAC.*

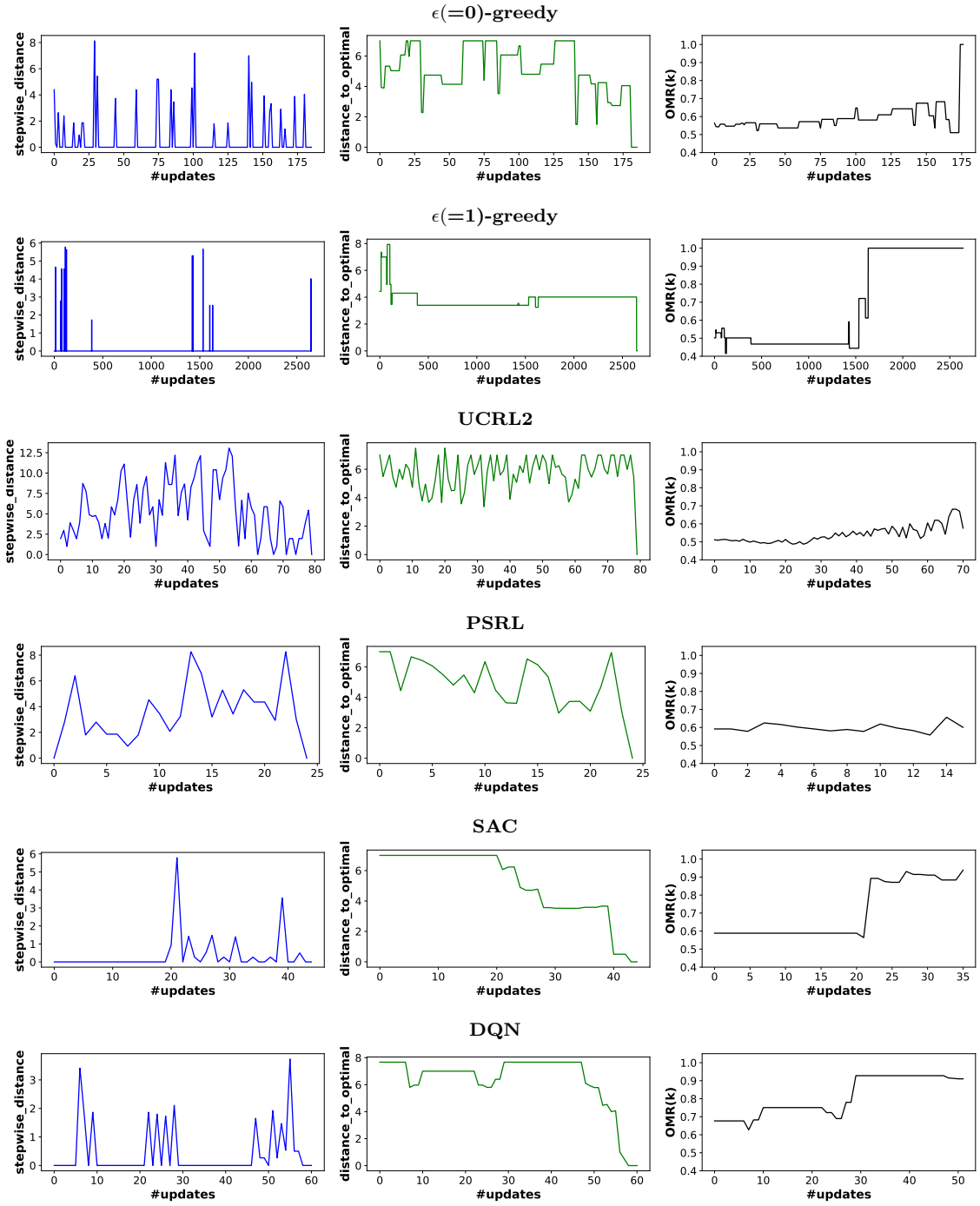


Figure B.7: *Plots in the first column are stepwise-distance vs. number of updates, second column distance-to-optimal vs. number of updates, and third OMR(k) vs. number of updates. The plots in the row belong to algorithms in the following order from top to bottom: $\epsilon(=0)$-greedy, $\epsilon(=1)$-greedy, UCRL2, PSRL, SAC, and DQN.*

B.5 Specifications of the RL Algorithms under Study

B.5.1 Methods for simulation results (Discrete MDP)

Model parameter initialisation. We initialised model parameters for deep learning RL algorithms like DQN and SAC by uniformly sampling weight values between $-3 \cdot 10^{-4}$ and $3 \cdot 10^{-4}$ and the biases at 0. For tabular Q-learning algorithms, we randomly initialized the Q-values between -1.0 and 1.0. For UCRL and PSRL, the policy model was randomly initialized. Note that all Wasserstein distances were computed using a python package POT (Flamary et al., 2021). Additionally, L1 norm was used in our Wasserstein metric cost function as the ground metric for the 2D gridworld environment.

Results in Figure 3.5. The problem setting was deterministic with dense-rewards and 15 maximum number of steps per episode. The total number of episodes was 200. The convergence criterion was satisfied when maximum returns were produced by an algorithm over 5 consecutive updates. The results showcase a single representative run of each algorithm. The confidence parameter $\delta = 0.1$ was utilized for UCRL2.

The α parameter for SAC was autotuned using the approach in (Haarnoja et al., 2019) along with hyperparameters described in Table B.3. While DQN began with $\epsilon = 1.0$ and the value decayed as $\epsilon[t+1] = \max\{0.9999 \times \epsilon[t], 0.0001\}$. Table B.4 shows hyperparameters for DQN. Note that the ADAM (Kingma & Ba, 2017) optimizer was used in all the neural network models.

Table B.3: *SAC Hyperparameters.*

Parameter	Value
learning rate	$5 \cdot 10^{-4}$
discount(γ)	0.99
replay buffer size	10^4
number of hidden layers (all networks)	1
number of hidden units per layer	32
number of samples per minibatch	64
nonlinearity	ReLU
entropy target	-4
target smoothing coefficient (τ)	0.01
target update interval	1
gradient steps	1
initial exploration steps before model starts updating	500

Results in Tables 3.1 and 3.2. The problem settings had 40 maximum number of steps per episode, and the convergence criterion was satisfied when maximum re-

Table B.4: *DQN Hyperparameters.*

Parameter	Value
learning rate	$5 \cdot 10^{-2}$
discount(γ)	0.99
replay buffer size	10^4
number of hidden layers (all networks)	1
number of hidden units per layer	32
number of samples per minibatch	64
nonlinearity	ReLU
target smoothing coefficient (τ)	0.001
target update interval	1
gradient steps	1
initial exploration steps before ϵ decays	500

turns were produced by an algorithm over 5 consecutive updates. The means and standard deviations for each algorithm were computed over 50 runs. The total number of episodes was 200 for results in Table 3.1, and 500 in Table 3.2. For results in Figure 3.8, the Q-learning with decaying ϵ-greedy where $\epsilon = 0.9$ was employed in the gridworld tasks. A convergence criterion of 50 consecutive model updates with maximum returns was utilized. We aggregated the result over 40 training trials and the maximum number of steps per episode was 60.

B.5.2 Methods for simulation results (Continuous MDP)

Model parameter initialisation. We initialised model parameters for the deep learning SAC algorithm by uniformly sampling weight values between $-3 \cdot 10^{-4}$ and $3 \cdot 10^{-4}$ and the biases at 0. For the DDPG algorithm, the output layer weight values were initialised using Xavier Initialization (Glorot & Bengio, 2010), while the rest were uniformly sampled between $-3 \cdot 10^{-3}$ and $3 \cdot 10^{-3}$. This was done on both the actor and critic networks. The ADAM (Kingma & Ba, 2017) optimizer was used in all the neural network models. In both algorithms, 1) a discount factor $\gamma = 0.99$ was used, 2) 500 initial steps were taken before updating model weights, and 3) replay buffer size was 10^6. Tables B.5 and C.1 display hyperparameters for DDPG and SAC, respectively.

Results in Figure 3.6. The problem setting was Mountain Car continuous (Moore, 1990) with 999 maximum number of steps per episode (Brockman et al., 2016). The total number of training episodes was 100. The convergence criterion was satisfied when maximum returns were produced by an algorithm over 10 consecutive updates. The results showcase a single representative run of each algorithm. For results in **Table 3.3**, the mean and standard deviations for each algorithm were computed over 5 runs.

While RL training was conducted in a continuous state-action space, we discretized it for Wasserstein distance calculations between occupancy measures, using 4 bins for actions and 10 bins for states. Note that all Wasserstein distances were computed

using a python package POT (Flamary et al., 2021). Additionally, L2 norm was used in our Wasserstein metric cost function as the ground metric for the Mountain Car environment.

Table B.5: *DDPG Hyperparameters.*

Parameter	Value
number of samples per minibatch	128
nonlinearity	ReLU
target smoothing coefficients (τ)	0.001
target update interval	1
gradient steps	1
number of hidden layers (all networks)	2
number of hidden units per layer	64
Actor learning rate	$5 \cdot 10^{-4}$
Critic learning rate	$5 \cdot 10^{-3}$

Table B.6: *SAC Hyperparameters (in the continuous setting).*

Parameter	Value
learning rate	$3 \cdot 10^{-3}$
number of hidden layers (all networks)	2
number of hidden units per layer	64
number of samples per minibatch	128
nonlinearity	ReLU
target smoothing coefficient (τ)	0.001
target update interval	1
gradient steps	1

Appendix C

Task Complexity Analysis

This section contains additional supporting material for Chapter 4.

C.1 Positional Errors at End-effector

This section demonstrates how the error in the joints propagates to the end-effector. Given a planar serial chain with n joints, the relationship between the end-effector velocities $\dot{x} \in \mathbb{R}^2$ and joint velocities $\dot{\theta} \in \mathbb{R}^n$ is (Sciavicco & Siciliano, 2012):

$$\dot{x} = J(\theta)\dot{\theta} \tag{C.1}$$

where $J(\theta) \in \mathbb{R}^{2\times n}$ is the Jacobian matrix. The $k-$th column of $J(\theta)$ is given by:

$$\begin{aligned} J_k(\theta) &= \frac{\partial x}{\partial \theta_k} \\ &= \begin{bmatrix} -\sum_{i=k}^{n} l_i \sin\left(\sum_{j=1}^{i} \theta_j\right) \\ \sum_{i=k}^{n} l_i \cos\left(\sum_{j=1}^{i} \theta_j\right) \end{bmatrix} \end{aligned} \tag{C.2}$$

whose Euclidean norm is

$$\|J_k(\theta)\|_2 = \sqrt{\left(\sum_{i=k}^{n} l_i \sin\left(\sum_{j=1}^{i} \theta_j\right)\right)^2 + \left(\sum_{i=k}^{n} l_i \cos\left(\sum_{j=1}^{i} \theta_j\right)\right)^2}$$

$$
\begin{aligned}
&= \sqrt{\sum_{i=k}^{n} l_i^2 + 2 \sum_{k \le i < h \le n}^{n} l_i l_h \cos\left(\sum_{j=1}^{i} \theta_j - \sum_{j=1}^{h} \theta_j\right)} \\
&\le \sum_{i=k}^{n} l_i
\end{aligned}
\tag{C.3}
$$

Given a small joint error, we can use Equation C.1 to estimate the error at the end-effector as:

$$
\begin{aligned}
\delta x &\approx J(\theta)\delta\theta \\
&= \sum_{k=1}^{n} J_k \delta\theta_k
\end{aligned}
\tag{C.4}
$$

Therefore, the magnitude of the $||\delta x||_2$ can be bounded using the triangular inequality as follows:

$$
\begin{aligned}
\|\delta x\|_2 &= \left\| \sum_{k=1}^{n} J_k \delta\theta_k \right\|_2 \\
&\le \sum_{k=1}^{n} \|J_k \delta\theta_k\|_2 \\
&= \sum_{k=1}^{n} \|J_k\|_2 \, |\delta\theta_k|
\end{aligned}
\tag{C.5}
$$

If $|\delta\theta_k| \le \epsilon$ for $k = 1, \cdots, n$, and we substitute Equation C.3 into Equation C.5, then

$$
\begin{aligned}
\|\delta x\|_2 &\le \sum_{k=1}^{n} \|J_k\|_2 \, \epsilon \\
&\le \epsilon \sum_{k=1}^{n} \sum_{i=k}^{n} l_i \\
&= \epsilon \sum_{i=1}^{n} \sum_{k=1}^{i} l_i \\
&= \epsilon \sum_{i=1}^{n} i l_i
\end{aligned}
\tag{C.6}
$$

The final results of Equation C.6 demonstrate that the error magnitude at the end-effector is exacerbated by both the number of joints n and link lengths l_i. Therefore,

the task complexity should increase when degree of freedom n increase and/or when the link lengths l_i increases, since an RL agent has to deal with increased error sensitivity.

C.2 SAC Architecture

In this section, we specify the architecture of the SAC algorithm used to produce the learning performances in Figure 4.2. Table C.1 entails a list of the configuration parameters for the algorithm. The ADAM (Kingma & Ba, 2017) optimiser was used in both the actor and critic (neural-network) models.

Table C.1: *SAC Hyperparameters.*

Parameter	Value
learning rate	0.001
discount factor (γ)	0.99
temperature coefficient (α)	0.2
number of hidden layers (all networks)	2
number of hidden units per layer	256
number of samples per minibatch	64
nonlinearity	ReLU
target smoothing coefficient (τ)	0.005
Experience Replay size	10^6

C.3 Estimation Methods

C.3.1 PIC and POIC values

This section describes how the mean and standard deviation of the PIC and POIC values presented in Table 4.1 were determined. We used bootstrap resampling of the 10^4 samples to estimate the distribution of PIC and POIC. This allowed us in addition to compute the confidence intervals shown in Table C.2.

C.3.2 Performance Distribution Plots

This section presents performance distribution plots (in Figure C.1) for the tasks *1-link (L=1.0m)* and *1-link (L=1.65m)* arms with sparse rewards.

C.3.3 Initialisation Methods

In RWG, we mainly used multivariate normal distribution prior $p(\theta) = \mathcal{N}(0, I)$, where $I \in \mathbb{R}^{d\times d}$ given weight vectors $\theta_n \in \mathbb{R}^d$, to sample weights $\theta_n \sim p(\theta)$. This was combined with a neural network architecture (NN) of 2 hidden layers (HL), each with 32 units (HU) and the NN is without bias, i.e. NN = [2HL, 32HU, w/o bias]. In Figure C.2, we show that performance distributions for the tasks are unchanging when prior distribution is altered. We tested with the following cases:

Table C.2: *Complete PIC and POIC values based on bootstrapping.*

Rewards	Tasks	Metric	Value	CI (95%)
Dense	*1-link* $(L = 1m)$	PIC	4.005 ± 0.0085	$(3.979; 4.012)$
Dense	*1-link* $(L = 1m)$	POIC	$(2.628 \pm 0.056) \cdot 10^{-3}$	$(2.520; 2.738) \cdot 10^{-3}$
Dense	*1-link* $(L = 1.65m)$	PIC	4.153 ± 0.0084	$(4.126; 4.159)$
Dense	*1-link* $(L = 1.65m)$	POIC	$(4.105 \pm 0.085) \cdot 10^{-3}$	$(3.944; 4.279) \cdot 10^{-3}$
Dense	*2-link*	PIC	4.205 ± 0.0061	$(4.195; 4.219)$
Dense	*2-link*	POIC	$(0.722 \pm 0.011) \cdot 10^{-3}$	$(0.701; 0.746) \cdot 10^{-3}$
Sparse	*1-link* $(L = 1m)$	PIC	0.0851 ± 0.0004	$(0.0843; 0.0859)$
Sparse	*1-link* $(L = 1m)$	POIC	$(1.982 \pm 0.034) \cdot 10^{-3}$	$(1.911; 2.047) \cdot 10^{-3}$
Sparse	*1-link* $(L = 1.65m)$	PIC	0.0713 ± 0.0002	$(0.0708; 0.0718)$
Sparse	*1-link* $(L = 1.65m)$	POIC	$(1.197 \pm 0.031) \cdot 10^{-3}$	$(1.106; 1.233) \cdot 10^{-3}$
Sparse	*2-link*	PIC	$0.0460 \pm 6.37 \cdot 10^{-5}$	$(0.0458; 0.0461)$
Sparse	*2-link*	POIC	$(0.946 \pm 0.0079) \cdot 10^{-3}$	$(0.932; 0.962) \cdot 10^{-3}$

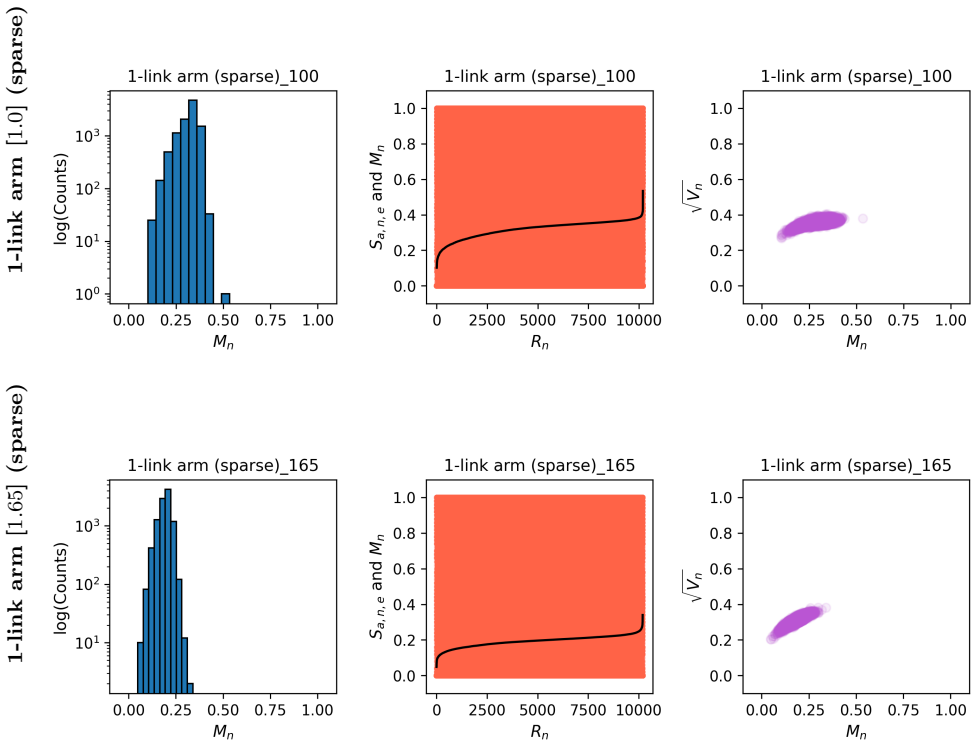


Figure C.1: *Performance distribution plots for the tasks 1-link (L=1.0m) and 1-link (L=1.65m) arms with sparse rewards. The performance is normalised using min-max scaling to allow tasks to have the same range.*

1. Default: $p(\theta) = \mathcal{N}(0, I)$ + NN = [2HL, 32HU, w/o bias]
2. w/ Bias: $p(\theta) = \mathcal{N}(0, I)$ + NN = [2HL, 32HU, w/ bias]
3. Variance: $p(\theta) = \mathcal{N}(0, 2I)$ + NN = [2HL, 32HU, w/o bias]

4. Units64: $p(\theta) = \mathcal{N}(0, I)$ + NN = [2HL, 64HU, w/o bias]
5. Units256: $p(\theta) = \mathcal{N}(0, I)$ + NN = [2HL, 256HU, w/ bias]
6. Uniform: $p(\theta) = Unif(-1, 1)$ + NN = [2HL, 32HU, w/o bias]

To attain the results in Figure C.2, we used $N = 10^3$ samples (i.e. random policies) in each setting, where each sample was deploy across 500 episodes.

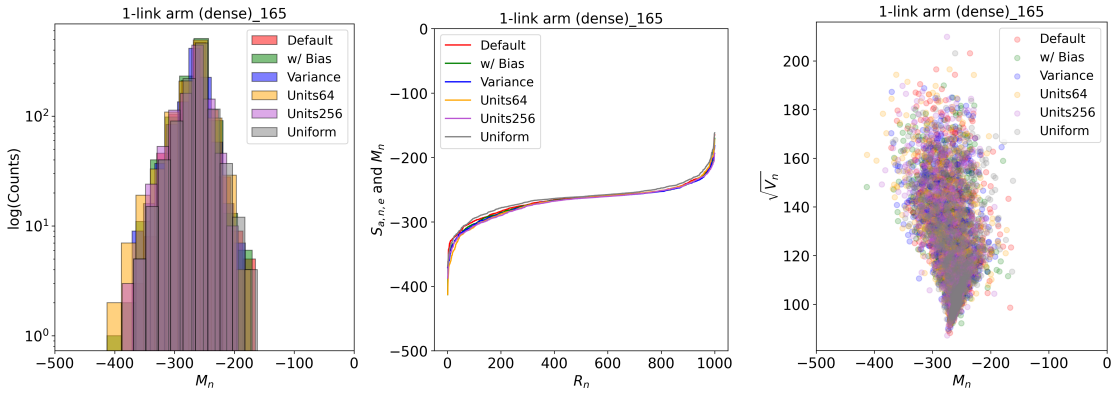
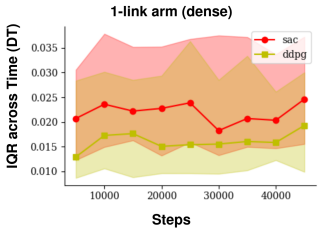


Figure C.2: *Performance distribution plots for the 1-link ($L = 1.65$ m) arm with dense rewards across various policy network architectures and initialisation methods of weights. This shows how the random policies behave similarly across various settings.*

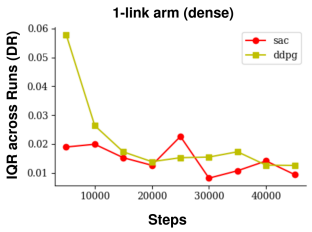
Appendix D

Reliability Metrics Results

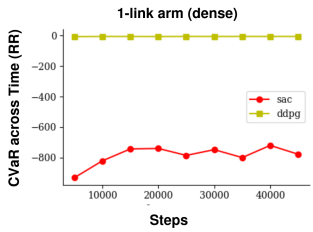
This section entails results of reliability analysis for the remaining tasks. Results are shown as follows: Figure D.1 for *1-link* arm task with dense-rewards, Figure D.2 for *2-link* arm task with dense-rewards, Figure D.4 for *2-link* arm task with sparse-rewards, and Figure D.6 for *UR10* arm task with sparse-rewards. For all the figures, statistically significant differences in ranking will have a black horizontal line above the bars, its absence indicates non-significance.



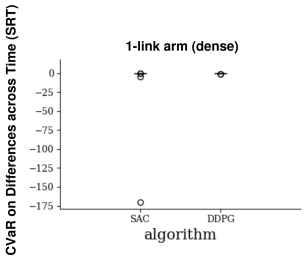
(a) *Dispersion across Time (Raw)*



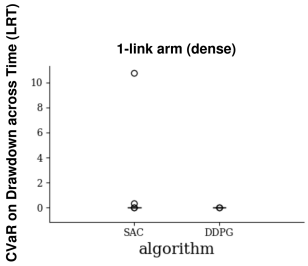
(b) *Dispersion across Runs (Raw) - for a representative run*



(c) *Risk across Runs (Raw) - for a representative run*

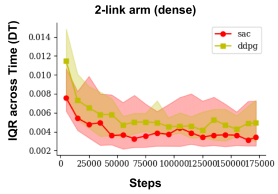


(d) *Short-term Risk across Time (Raw)*



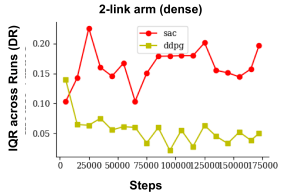
(e) *Long-term Risk across Time (Raw)*

Figure D.1: *Reliability metrics rankings for the **1-link arm task with dense rewards** during training. The figure displays raw metric values of dispersion across time (DT), dispersion across runs (DR), risk of worst performance (with 5% probability) across runs (RR), box plot of risk of performance drop with 5% probability over short-terms during the runs (SRT), and risk of performance drop with 5% probability across long-terms during the runs (LRT). Note that less positive values in raw metrics for figures (a), (b), (c) and (e) represent better outcome. While more positive values in figure (d) denote better risk.*



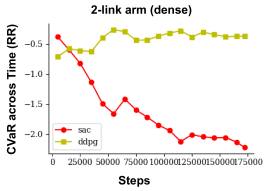
(a) *Dispersion across Time (Raw)*

(b) *Dispersion across Time (Ranking per timeframe)*

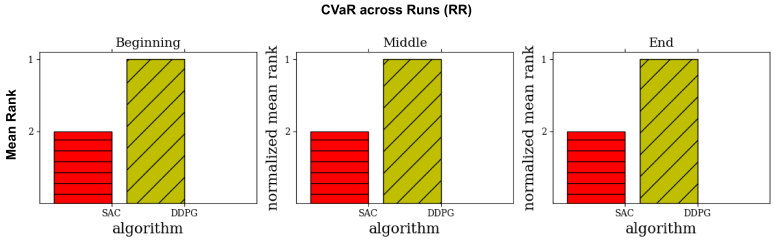


(c) *Dispersion across Runs (Raw) - for a representative run*

(d) *Dispersion across Runs (Ranking per timeframe)*

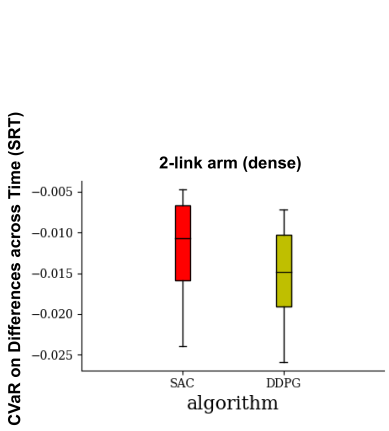


(e) *Risk across Runs (Raw) - for a representative run*

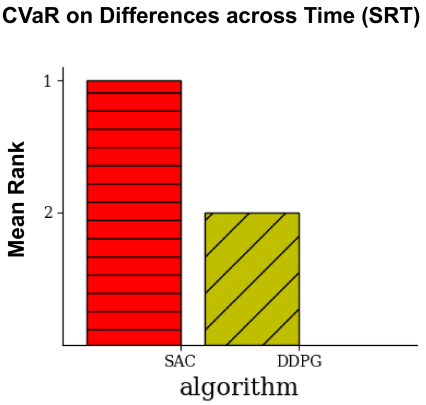


(f) *Risk across Runs (Ranking per timeframe)*

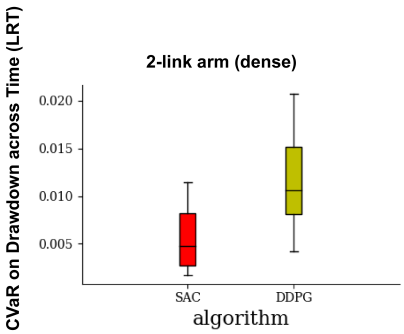
Figure D.2: *Reliability metrics rankings for the **2-link arm task with dense rewards** during training. Note that rank 1 always indicates better metric outcome. (a), (c) and (e) respectively illustrate raw metric values of dispersion across time, dispersion across runs, and risk of worst performance (with 5% probability) across runs. (b), (d) and (f) are corresponding metric rankings per timeframe during training. Note that less positive values in raw metrics represent better outcome.*



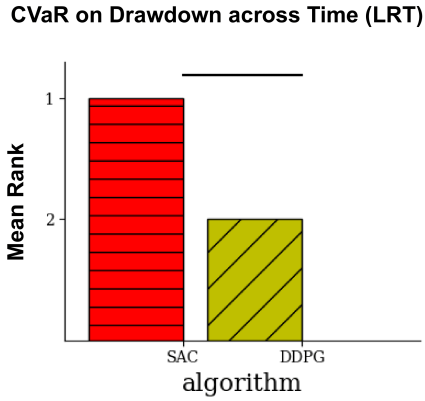
(a) *Short-term Risk across Time (Raw)*



(b) *Short-term Risk across Time (Ranking)*

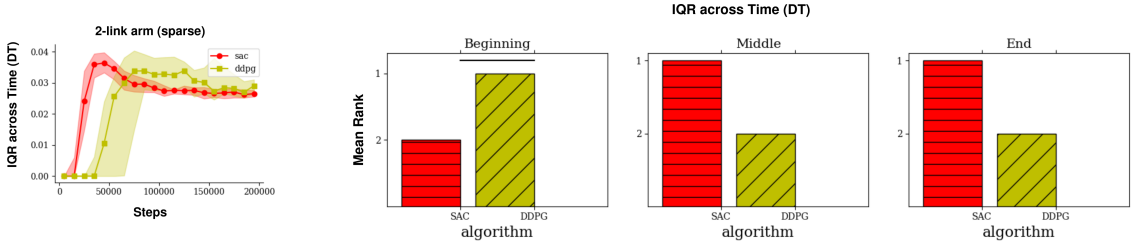


(c) *Long-term Risk across Time (Raw)*



(d) *Long-term Risk across Time (Ranking)*

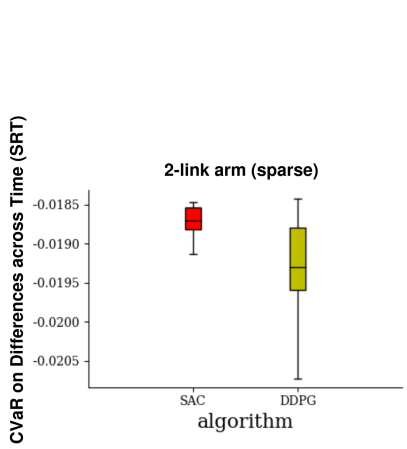
Figure D.3: *Reliability metrics ranking for the **2-link arm task with dense rewards** during training. Rank 1 indicates better metric outcome. (a) shows box plots of risk of performance drop with 5% probability over short-terms during the runs, while (b) depicts the corresponding mean rankings. (c) portrays risk of performance drop with 5% probability across long-terms during the runs, while (d) outlines the corresponding mean rankings. Note that more positive values in (a) denote better risk, and in (c) less positive values are superior.*



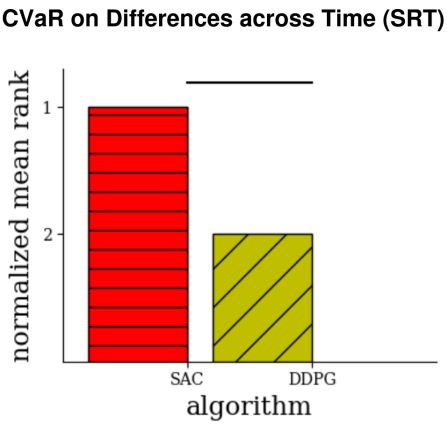
(a) *Dispersion across Time (Raw)*

(b) *Dispersion across Time (Ranking per timeframe)*

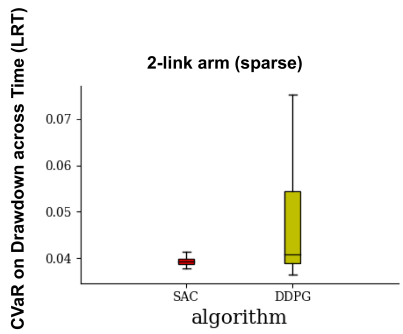
Figure D.4: *Reliability metrics rankings for the **2-link arm task with sparse rewards** during training. Note that rank 1 always indicates better metric outcome. (a) illustrates raw metric values of dispersion across time. (b) is the corresponding metric rankings per timeframe during training. Note that less positive values in raw metrics represent better outcome.*



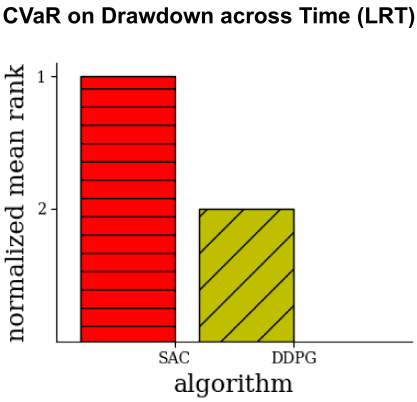
(a) *Short-term Risk across Time (Raw)*



(b) *Short-term Risk across Time (Ranking)*

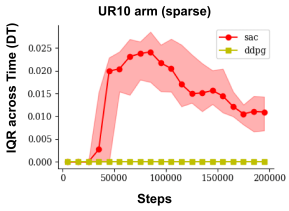


(c) *Long-term Risk across Time (Raw)*

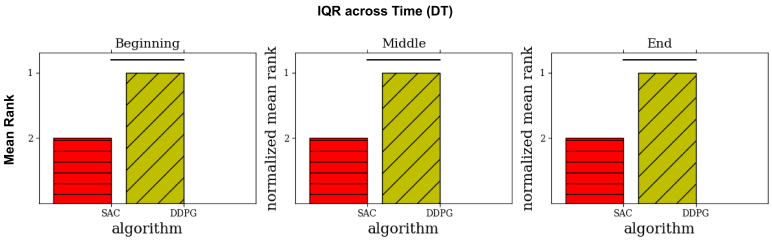


(d) *Long-term Risk across Time (Ranking)*

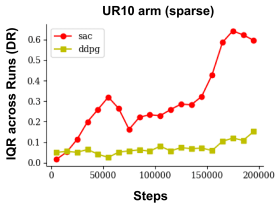
Figure D.5: *Reliability metrics ranking for the **2-link arm task with sparse rewards** during training. Rank 1 indicates better metric outcome. (a) shows box plots of risk of performance drop with 5% probability over short-terms during the runs, while (b) depicts the corresponding mean rankings. (c) portrays risk of performance drop with 5% probability across long-terms during the runs, while (d) outlines the corresponding mean rankings. Note that more positive values in (a) denote better risk, and in (c) less positive values are superior.*



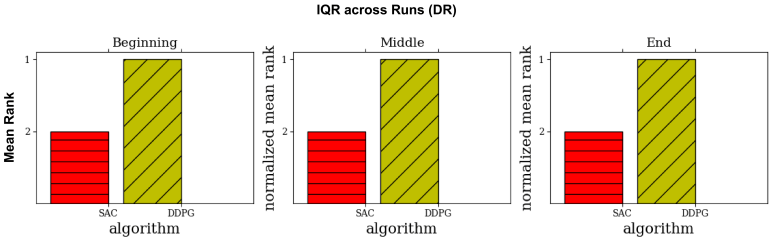
(a) *Dispersion across Time (Raw)*



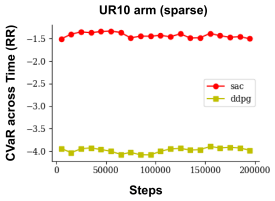
(b) *Dispersion across Time (Ranking per timeframe)*



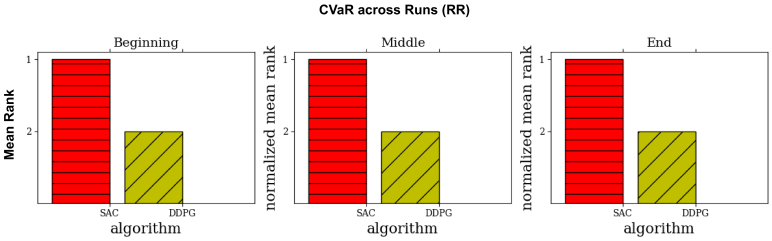
(c) *Dispersion across Runs (Raw) - for a representative run*



(d) *Dispersion across Runs (Ranking per timeframe)*



(e) *Risk across Runs (Raw) - for a representative run*



(f) *Risk across Runs (Ranking per timeframe)*

Figure D.6: *Reliability metrics rankings for the* ***UR10 arm task with sparse rewards*** *during training. Note that rank 1 always indicates better metric outcome. (a), (c) and (e) respectively illustrate raw metric values of dispersion across time, dispersion across runs, and risk of worst performance (with 5% probability) across runs. (b), (d) and (f) are corresponding metric rankings per timeframe during training. Note that less positive values in raw metrics represent better outcome.*

Appendix E

Hyperparameter Sensitivity Results

This section presents results of hyperparameter sensitivity for the remaining tasks. Results are depicted as follows: Figures E.1 and E.2 for *1-link* arm task with dense-rewards; Figures E.3 and E for *1-link* arm task with sparse-rewards. Figures E.5 and E.6 for *2-link* arm task with dense-rewards; Figures E.7 and E.8 for *2-link* arm task with sparse-rewards. Figures E.9 and E.10 for *UR10* arm task with sparse-rewards.

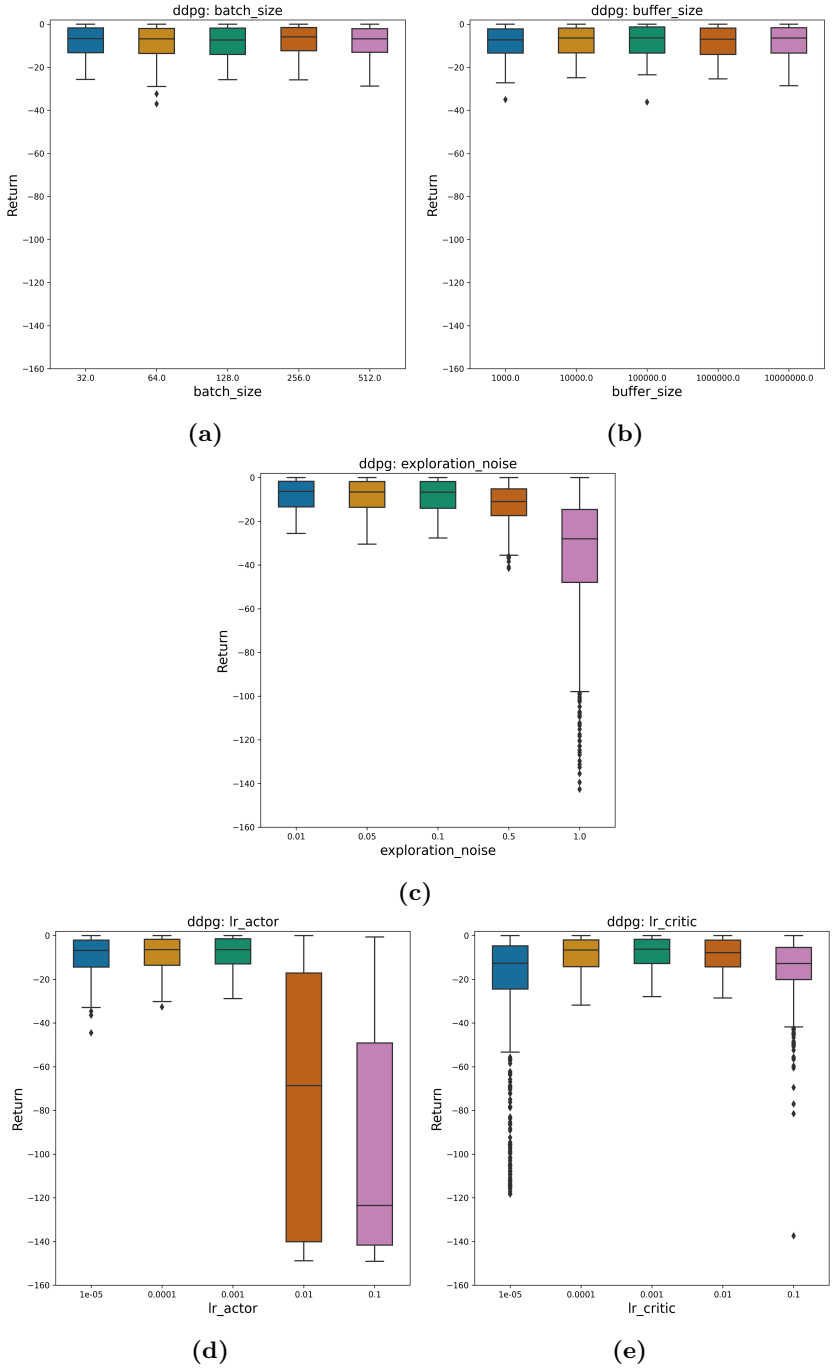


Figure E.1: *Performance of DDPG algorithm over a range of hyperparameter values for 1-link arm dense-reward task. In each plot, a single hyperparameter is varied while keeping others at their nominal values. Plots of performance variability due to batch size (a), buffer size (b), exploration noise (c), actor learning rate (d), and critic learning rate (e) are presented respectively.*

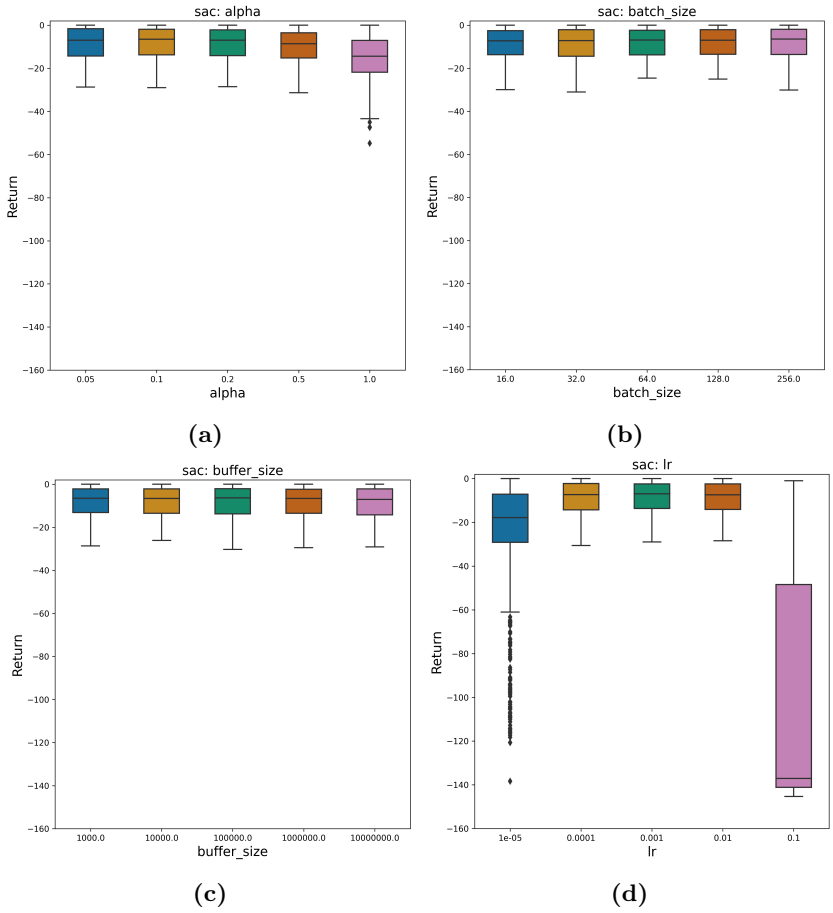


Figure E.2: *Performance of SAC algorithm over a range of hyperparameter values for 1-link arm dense-reward task. In each plot, a single hyperparameter is varied while keeping others at their nominal values. Plots of performance variability due to alpha (a), batch size (b), buffer size (c), and learning rate (d) are illustrated respectively.*

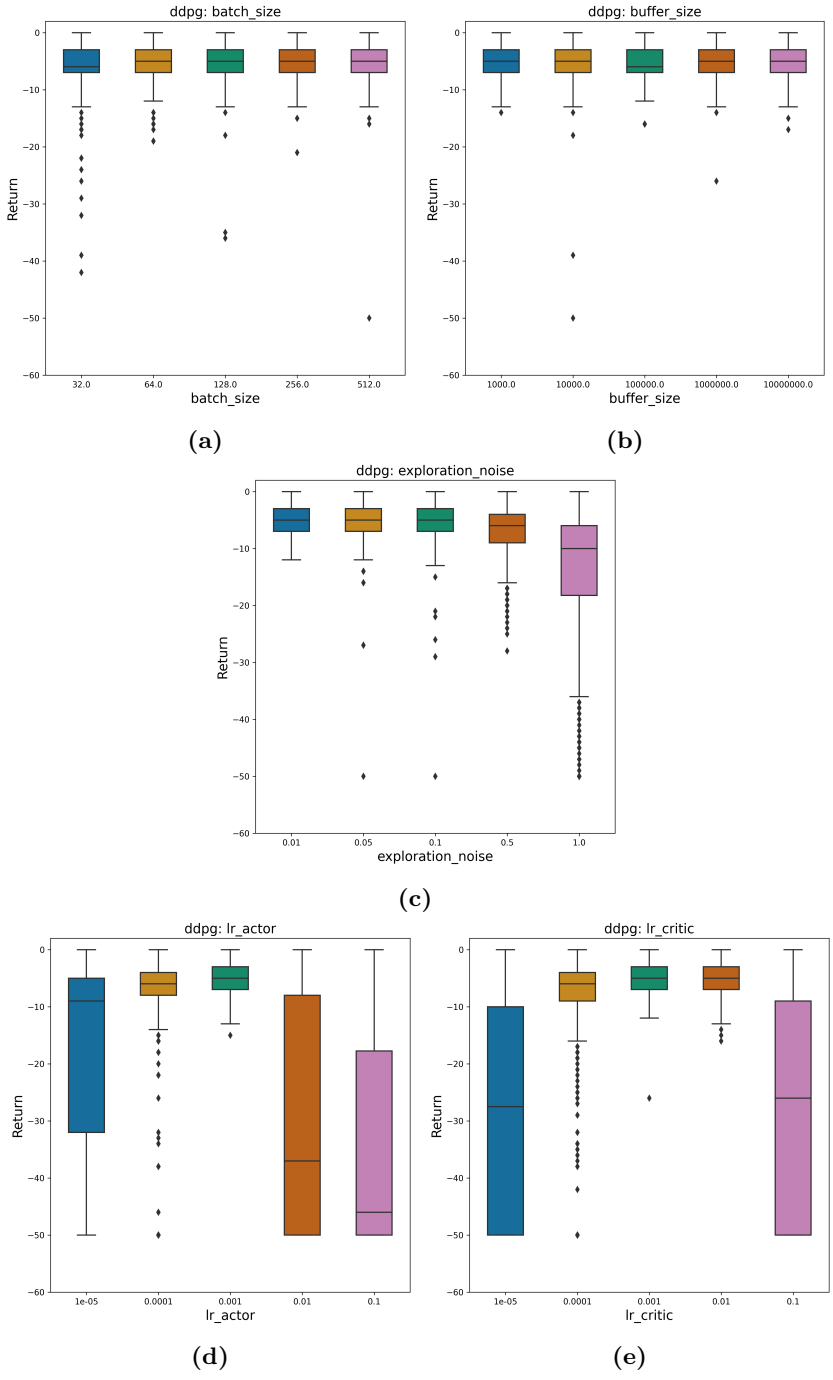


Figure E.3: *Performance of DDPG algorithm over a range of hyperparameter values for 1-link arm sparse-reward task. In each plot, a single hyperparameter is varied while keeping others at their nominal values. Plots of performance variability due to batch size (a), buffer size (b), exploration noise (c), actor learning rate (d), and critic learning rate (e) are presented respectively.*

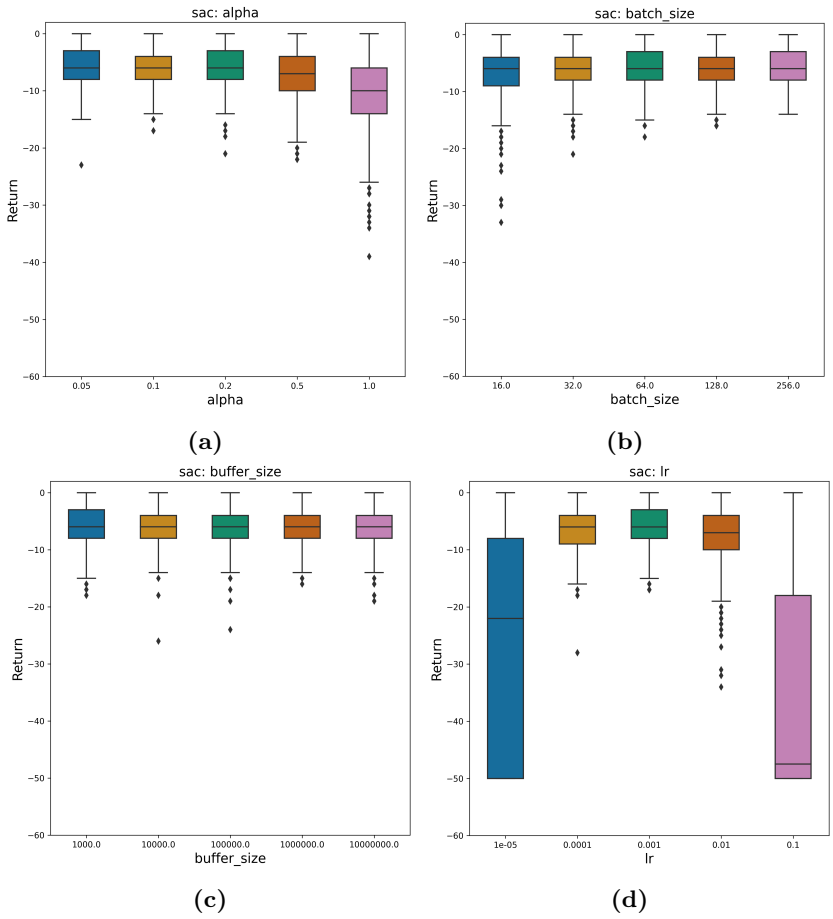


Figure E.4: *Performance of SAC algorithm over a range of hyperparameter values for 1-link arm sparse-reward task. In each plot, a single hyperparameter is varied while keeping others at their nominal values. Plots of performance variability due to alpha (a), batch size (b), buffer size (c), and learning rate (d) are illustrated respectively.*

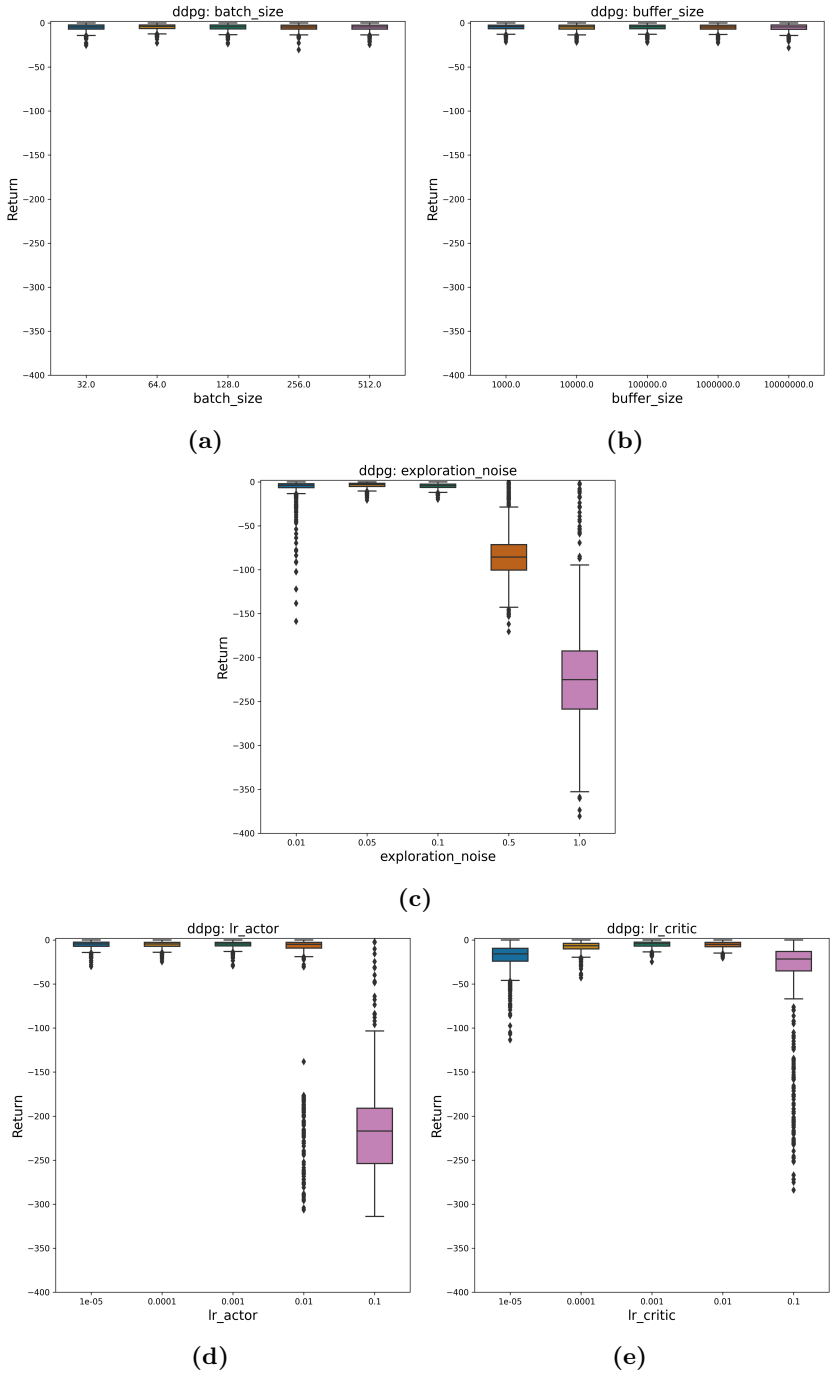


Figure E.5: *Performance of DDPG algorithm over a range of hyperparameter values for 2-link arm dense-reward task. In each plot, a single hyperparameter is varied while keeping others at their nominal values. Plots of performance variability due to batch size (a), buffer size (b), exploration noise (c), actor learning rate (d), and critic learning rate (e) are presented respectively.*

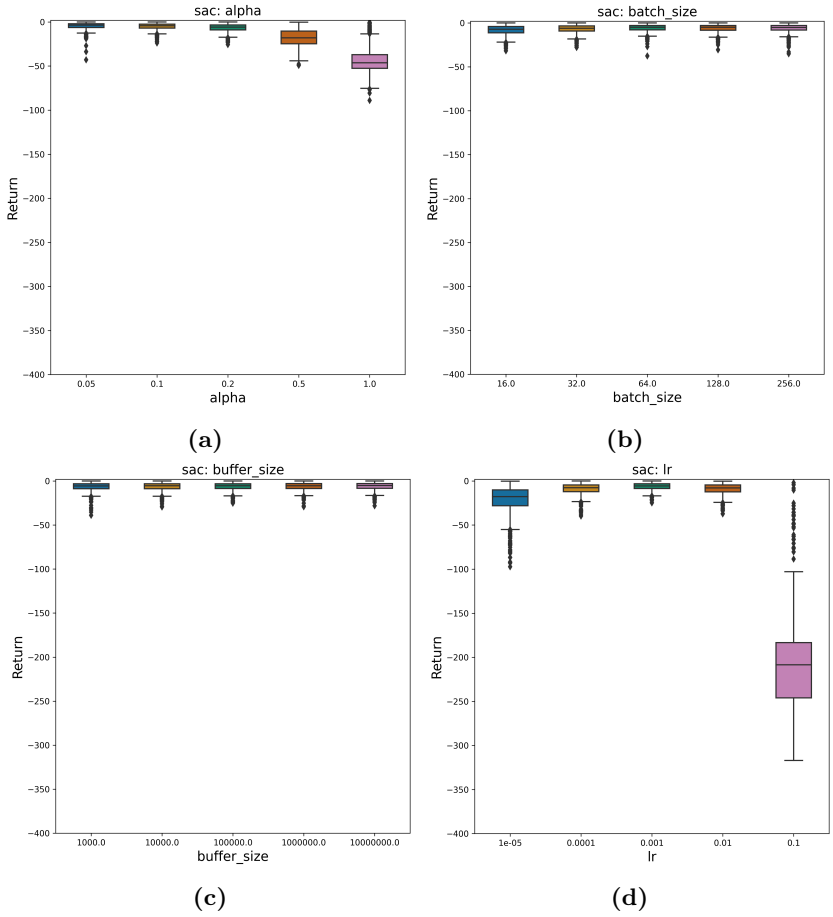


Figure E.6: *Performance of SAC algorithm over a range of hyperparameter values for 2-link arm dense-reward task. In each plot, a single hyperparameter is varied while keeping others at their nominal values. Plots of performance variability due to alpha (a), batch size (b), buffer size (c), and learning rate (d) are illustrated respectively.*

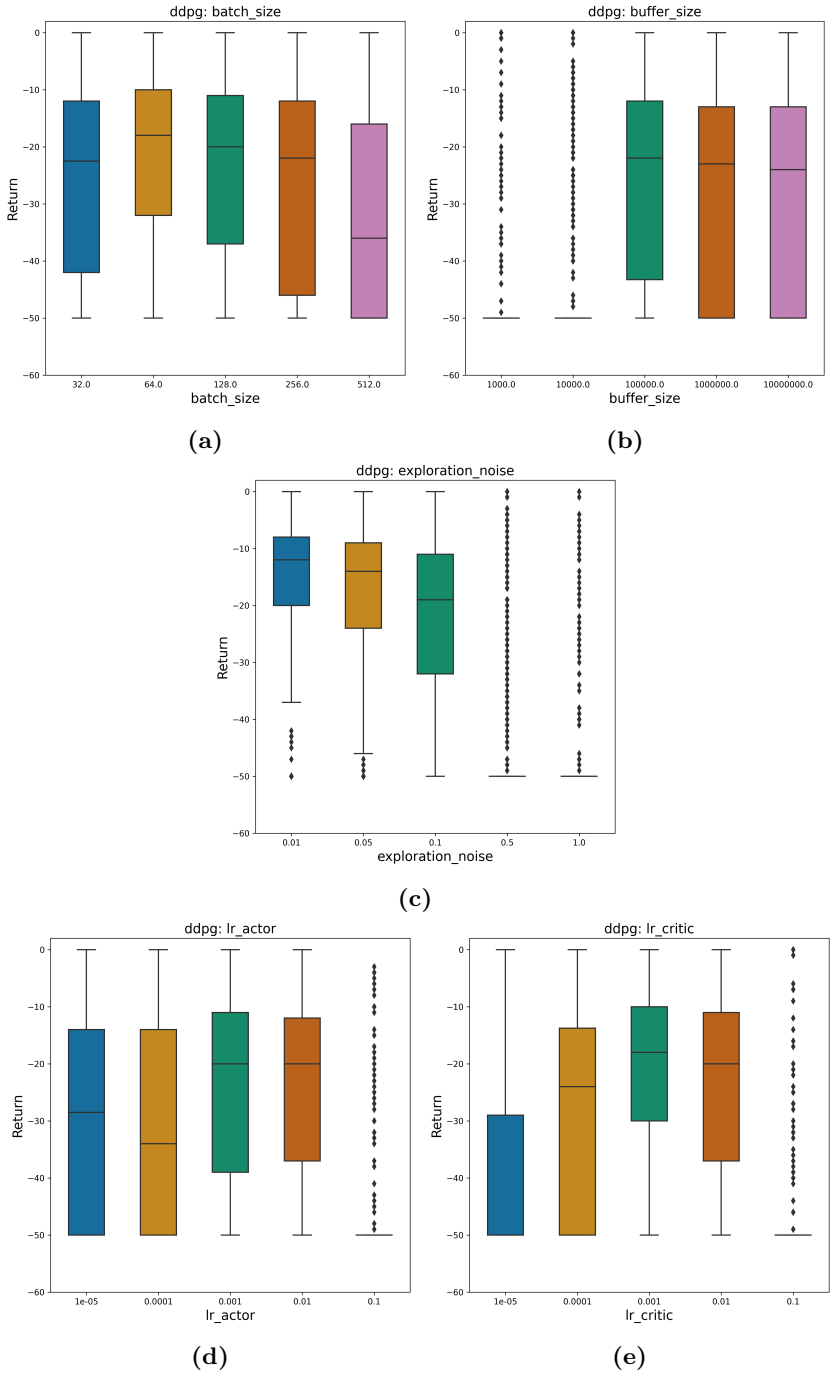


Figure E.7: *Performance of DDPG algorithm over a range of hyperparameter values for 2-link arm sparse-reward task. In each plot, a single hyperparameter is varied while keeping others at their nominal values. Plots of performance variability due to batch size (a), buffer size (b), exploration noise (c), actor learning rate (d), and critic learning rate (e) are presented respectively.*

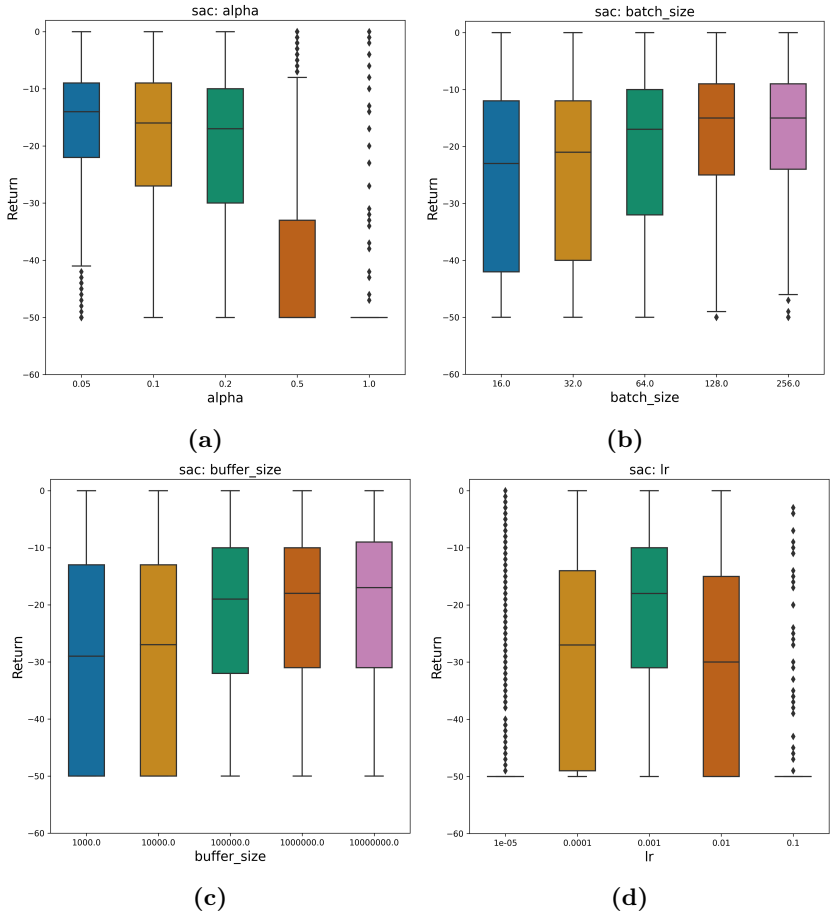


Figure E.8: *Performance of SAC algorithm over a range of hyperparameter values for 2-link arm sparse-reward task. In each plot, a single hyperparameter is varied while keeping others at their nominal values. Plots of performance variability due to alpha (a), batch size (b), buffer size (c), and learning rate (d) are illustrated respectively.*

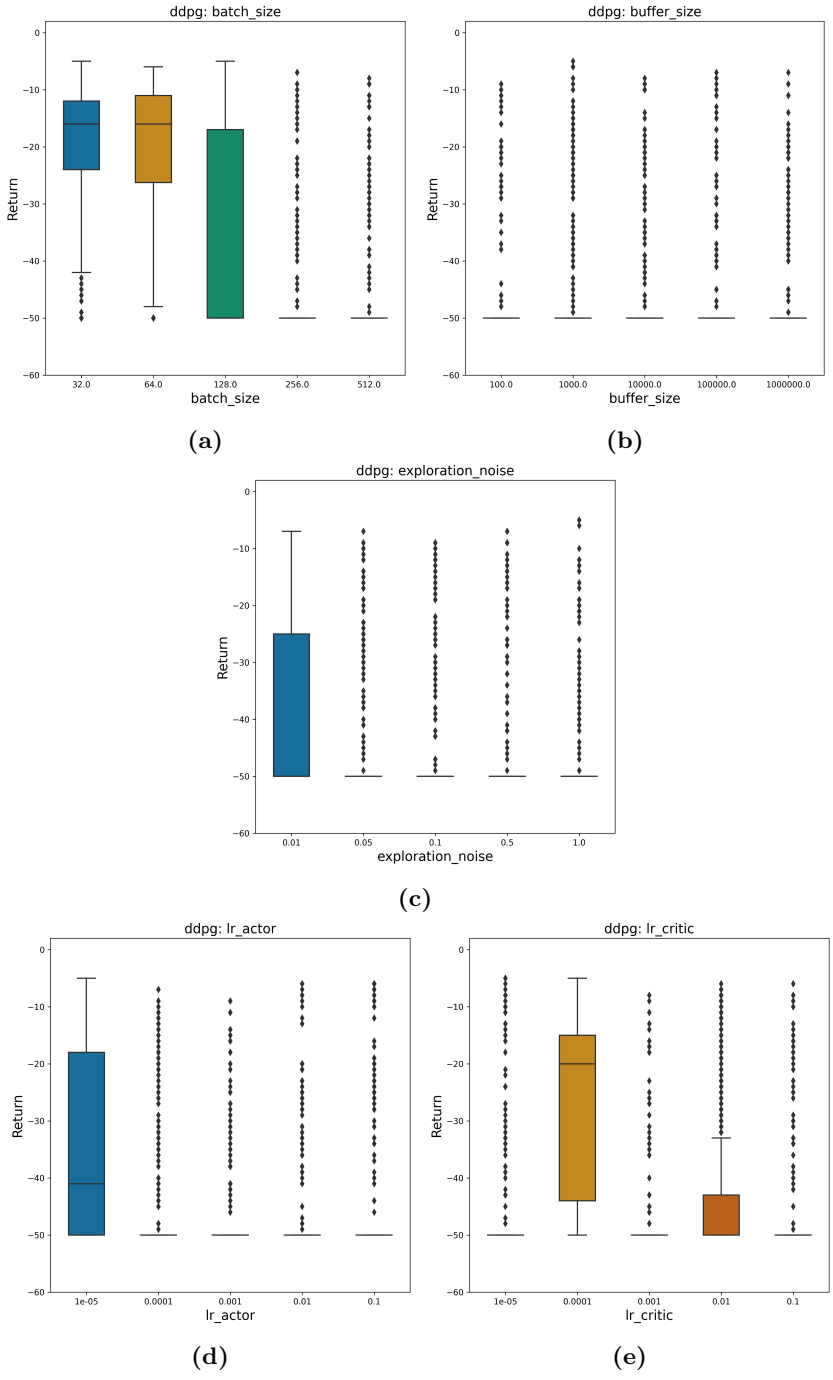


Figure E.9: *Performance of DDPG algorithm over a range of hyperparameter values for UR10 arm sparse-reward task. In each plot, a single hyperparameter is varied while keeping others at their nominal values. Plots of performance variability due to batch size (a), buffer size (b), exploration noise (c), actor learning rate (d), and critic learning rate (e) are presented respectively.*

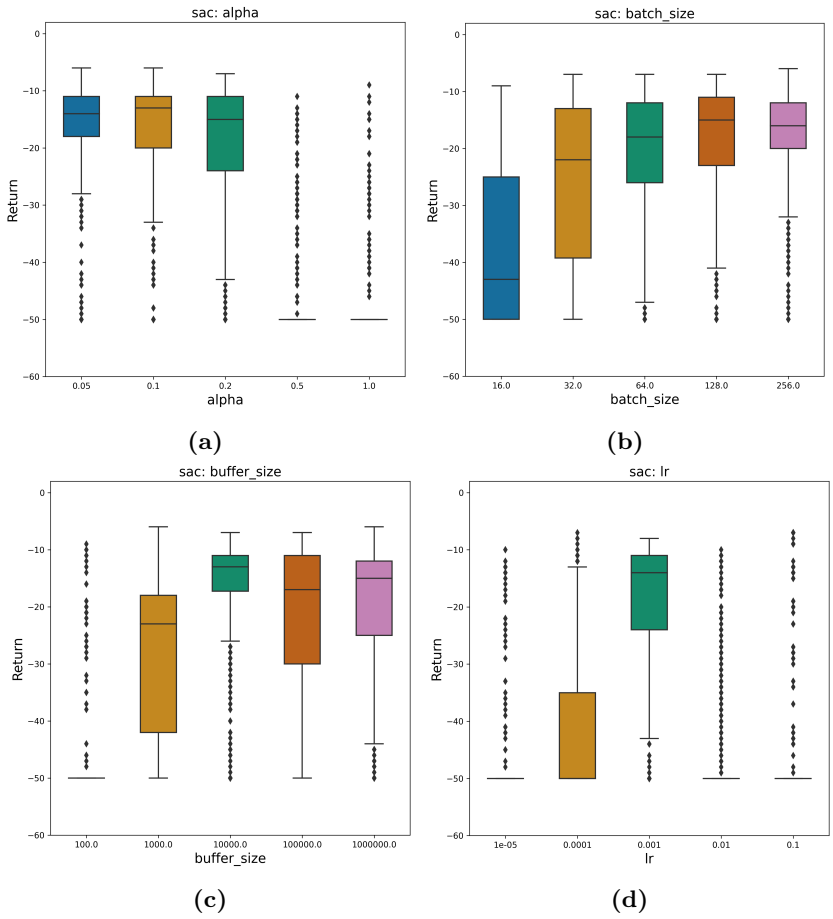


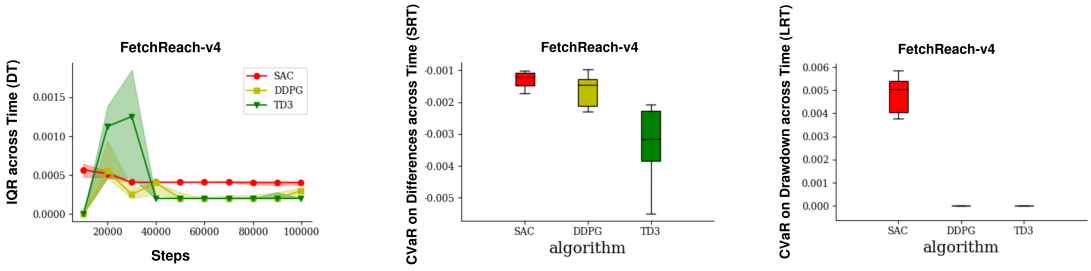
Figure E.10: *Performance of SAC algorithm over a range of hyperparameter values for UR10 arm sparse-reward task. In each plot, a single hyperparameter is varied while keeping others at their nominal values. Plots of performance variability due to alpha (a), batch size (b), buffer size (c), and learning rate (d) are illustrated respectively.*

Appendix F

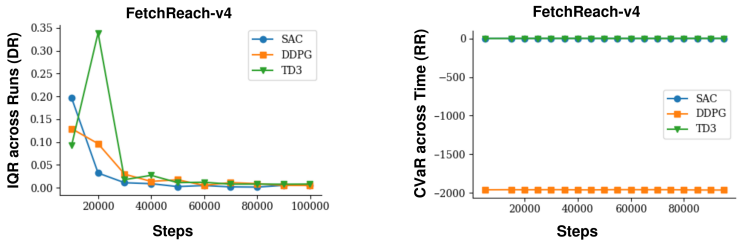
Multi-performance analysis

This section contains additional supporting material for Chapter 6. We begin with presenting raw results for Figure 6.3, followed by details about training the cVAEs in Section F.2, and finally, in Section F.3 we present a description of the underlying process behind Figure 6.4.

F.1 Raw Reliability Results



(a) *Dispersion across Time* **(b)** *Short-term Risk across Time* **(c)** *Long-term Risk across Time*



(d) *Dispersion across Runs* **(e)** *Risk across Runs*

Figure F.1: *Raw results for reliability metrics across algorithms SAC, DDPG and TD3 for the FetchReach-v4 task during training.*

F.2 Conditional variational autoencoders (cVAE)

The loss function cVAEs seek to minimise is (Higgins et al., 2017a; Alemi et al., 2018):

$$\mathcal{L} = \beta \cdot \mathrm{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{s}, \mathbf{a}) \mid p_\theta(\mathbf{z} \mid \mathbf{a})\right) - \mathbb{E}_{q_\phi}\left[\log p(\mathbf{s} \mid \mathbf{a}, \mathbf{z})\right] \tag{F.1}$$

where $\mathbf{z}$ is the latent variable and $\mathbf{s}$ is the observed variable (states) as conditioned on $\mathbf{a}$ (action). β is a hyperparameter that trades-off between the reconstruction accuracy and the closeness of the encoder q_ϕ to a normal distribution $p_\theta(\mathbf{z} \mid \mathbf{a}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ (i.e. latent space regularization) (Higgins et al., 2017a). θ and ϕ represent parameters of the encoder and decoder, respectively, as portrayed in Figure F.2.

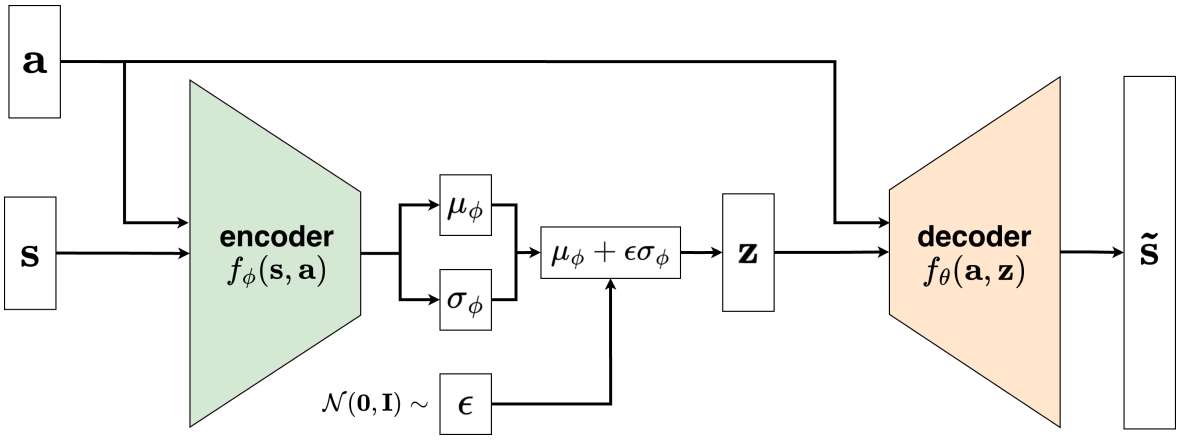
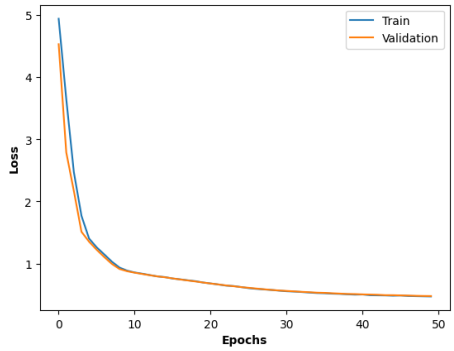


Figure F.2: *Schematic of the structure of a conditional variational autoencoder. The encoder that takes* $\mathbf{s}, \mathbf{a}$ *and produces parameters for a distribution* $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{s}, \mathbf{a}) = \mathcal{N}(\mu_\phi, \sigma_\phi)$ *over the latent space.* $\mathbf{z}$ *samples are decoded such that* $\tilde{\mathbf{s}} \sim p_\theta(\mathbf{s} \mid \mathbf{a}, \mathbf{z}) = f_\theta(\mathbf{a}, \mathbf{z})$.

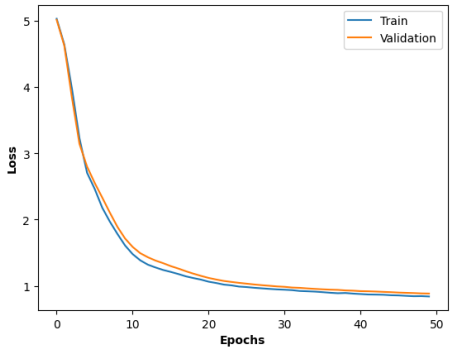
In our setting, the reconstruction accuracy is important as we want samples generated by the cVAE to have high fidelity. In contrast, the closeness of the encoder to the normal distribution is not so critical, since we do not use the encoder after training the cVAE. As part of assessing the soundness of samples generated by our cVAEs, we present the training and validation losses of representative cVAE models in Figure F.3. To construct Figure F.3, we worked with $\beta = 0.05$ and $\beta = 0.1$. The training samples were split 80%/20%, where the 80% was utilised for training and 20% for validation. Across all the epochs, the best loss value plateaued around < 0.8.

The training and validation losses consistently decrease over the epochs, thus highlighting an improvement in the fidelity of cVAE. Additionally, we performed k-fold cross validation (Wong & Yeh, 2020) to examine the variance of the validation losses during cVAE training. The results are delineated in Table F.1. We observe that the standard deviation values are significantly smaller than the mean values (less than 4%). This denotes that our cVAE models were not overfitting.

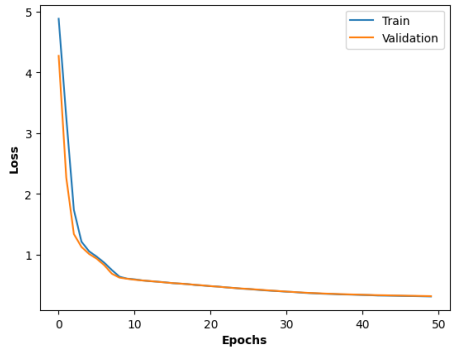
Another test we conducted was to compare the cVAE's sample distribution with the



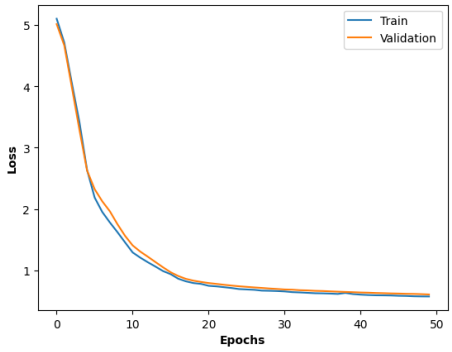
(a) *Loss Curve for SAC policy* π_0, $\beta = 0.1$



(b) *Loss Curve for SAC policy* π_{105}, $\beta = 0.1$



(c) *Loss Curve for SAC policy* π_0, $\beta = 0.05$



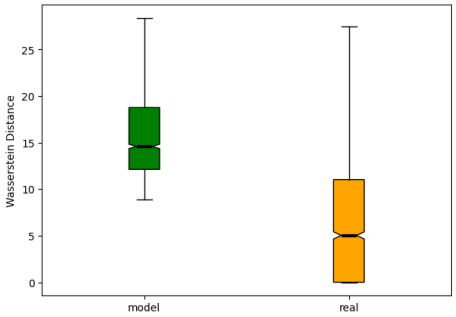
(d) *Loss Curve for SAC policy* π_{105}, $\beta = 0.05$

Figure F.3: *Training and validation loss curves for cVAE models for* $\beta = 0.05$ *and* $\beta = 0.1$. *The representative policy datasets selected are from SAC algorithm for policies* π_0 *and* π_{105} *during training.*

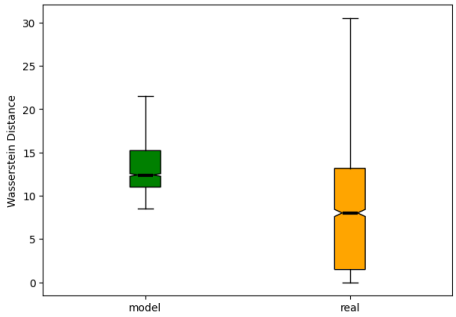
Table F.1: *K-cross validation with k*=10. *The representative policy datasets selected are from SAC algorithm for policies* π_0, π_{105} *and* $\pi_{200} = \pi^*$ *during training.*

β	Policy No.	mean	std ($\times 10^{-3}$)
0.05	0	0.2205	3.351
	105	0.4380	7.622
	200	0.4377	9.916
0.1	0	0.3538	11.40
	105	0.6883	11.63
	200	0.7132	13.13

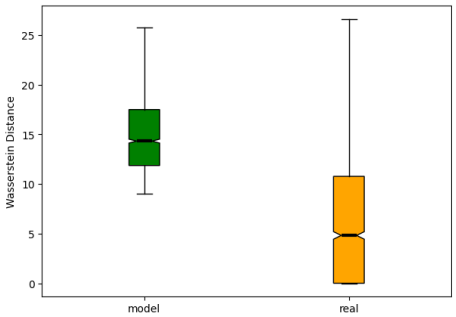
distribution of states in the neighbourhood of the state associated with the given conditional action. The underlying assumption here is that the states in the neighbourhood of the s (corresponding with action a) should not be far dissimilar with samples generated by a cVAE model. Following the procedure listed below, the results are depicted in Figure F.4.



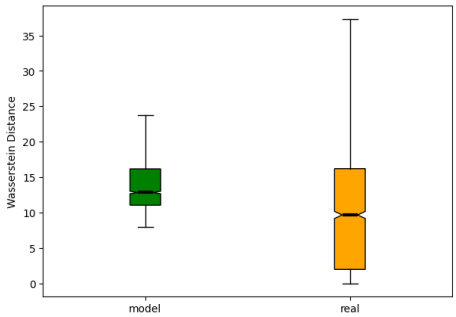
(a) *Wasserstein distance distribution with* $\beta = 0.1$ *and neighbourhood threshold* $\epsilon = 0.05$



(b) *Wasserstein distance distribution with* $\beta = 0.1$ *and neighbourhood threshold* $\epsilon = 0.1$



(c) *Wasserstein distance distribution with* $\beta = 0.05$ *and neighbourhood threshold* $\epsilon = 0.05$.



(d) *Wasserstein distance distribution with* $\beta = 0.05$ *and neighbourhood threshold* $\epsilon = 0.1$

Figure F.4: *cVAE's sample distribution for conditional action a vs distribution of states in the neighbourhood of state s which is associated the action a. The results are for a representative SAC policy* π_{105} *across various* β *values and neighbourhood threshold distances* ϵ.

Our procedure is a follows:

1. Randomly selected a representative set of conditioning points $\{a_1, a_2, \cdots, a_{M=2000}\}$ which are actions.

2. For each a_m, we collected a local real sample of states $Z_m = \{s_i \ : \parallel s_m - s_i \parallel \leq \epsilon\}$ from the policy Dataset, where $\epsilon = 0.1$ or 0.05. It is assumed that state samples in Z_m are the ground truth, since actions in close proximity should have similar states.

3. We then generated $M = 2000$ samples from the cVAE model $W_m = \{\hat{s}_i \sim \hat{p}(\cdot \mid a_m)\}_{m=1}^{M}$.

4. Then the discrepancy between these two sets of state samples was computed using 2-Wasserstein distance $\mathcal{W}_2(Z_m, W_m)$ over M conditioning points.

5. We collected the resulting Wasserstein distances $\{\mathcal{W}_2(Z_1, W_1), \cdots, \mathcal{W}_2(Z_M, W_M)\}$ and used them to construct the green boxplot (labelled *model*) portrayed in Figure F.4. This is a distribution denoting how far apart the samples are.

6. To form a baseline, for each a_m in $\{a_1, a_2, \cdots, a_{M=2000}\}$, we took its corresponding s_m — in the original dataset (policy dataset) $\mathcal{D}_\pi$ — and determined the Wasserstein distance $\mathcal{W}_2(s_m, W_m)$. This is the Wasserstein distance between the reference sample s_m and its neighbours.

7. We collected the resulting Wasserstein distances $\{\mathcal{W}_2(s_1, W_1), \cdots, \mathcal{W}_2(s_M, W_M)\}$ and constructed the orange boxplot labelled *real*.

By comparing the *model* distance distribution with the *real* distance distribution in Figure F.4, we notice an overlap and for $\epsilon = 0.1$, the medians are close. These results along with the loss curves signify that the samples from the cVAE are reasonably sound.

F.3 Exploration Analysis Results

The minimum number of policy model updates across all the algorithms was $20,000$. Having to visualise the policy evolution for more than $20,000$ updates was computationally expensive and too noisy. So, we introduced smoothening by selecting about 40 equally-spaced update points, which we utilised solely for visualisation in Figure 6.4. The final policy model update numbers are listed in Table F.2.

Table F.2: *Evaluation of RL algorithms in the FetchReach-v4 environment (over 5 runs). Note that* ***# updates*** *denotes the number of updates to converge, and values in* ***bold*** *signify best metric performance.*

Algo.	ESL	OMR	# updates ($\times 10^3$)
DDPG	34.15±2.93	0.514±0.05	36
SAC	**27.06±3.21**	**0.559±0.07**	**20**
TD3	30.73±0.96	0.528±0.07	30