# Robustness of Reinforcement Learning Systems in Real-World Environments

By

Juan José Garau Luis

B.S., Telecommunications Engineering, UPC BarcelonaTech, 2017
B.S., Industrial Engineering, UPC BarcelonaTech, 2017
S.M., Massachusetts Institute of Technology, 2020

Submitted to the Department of Aeronautics and Astronautics
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY IN AERONAUTICS AND ASTRONAUTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

July 2023



| | |
|---|---|
| Authored by: | Juan José Garau Luis<br>Department of Aeronautics and Astronautics<br>July 18, 2023 |
| Certified by: | Edward F. Crawley<br>Ford Professor of Engineering, MIT<br>Thesis Committee Chair |
| Certified by: | Cathy Wu<br>Professor of Civil and Environmental Engineering, MIT<br>Thesis Committee Member |
| Certified by: | Aleksandra Faust<br>Senior Staff Research Scientist, Google<br>Thesis Committee Member |
| Accepted by: | Jonathan P. How<br>R. C. Maclaurin Professor of Aeronautics and Astronautics<br>Graduate Program Committee Chair |

# Robustness of Reinforcement Learning Systems in Real-World Environments

by

Juan José Garau Luis

Submitted to the Department of Aeronautics and Astronautics
on July 18, 2023, in partial fulfillment of the
requirements for the Degree of
DOCTOR OF PHILOSOPHY IN AERONAUTICS AND ASTRONAUTICS

## Abstract

Reinforcement Learning (RL) is recognized as a promising paradigm to improve numerous decision-making processes in the real world, potentially constituting the core of many future autonomous systems. However, despite its popularity across multiple fields, the number of proofs of concept in the literature is substantially larger than the number of reported deployments. This can be primarily attributed to differences between real-world environments and experimental RL setups. On one hand, from a domain-specific perspective, it is challenging to fully characterize concrete tasks and environments in the real world, and training in physical environments may not always be possible. On the other hand, the real world presents several domain-agnostic challenges that make learning more difficult, such as high-dimensionality, non-stationarity, or generalizability. Although RL agents have demonstrated effective performance in practical applications, their robustness to these real-world phenomena is still challenging.

To move a step forward towards better RL deployability, this thesis investigates different aspects of RL system design, focusing on enhancing robustness in real-world environments. It is composed of three main areas of research:

Firstly, to comprehensively characterize the problem of real-world robustness, I propose an RL roadmap. This identifies key factors that influence the interaction between an RL system and a real-world environment, and it offers a structured approach to addressing the overall problem. I further delve into one specific element of this roadmap, the state space, and present a set of mathematical bounds for the change in mutual information (MI) between state features and rewards during policy learning. By observing how MI evolves during learning, I demonstrate how to identify more effective feature sets, as shown through the study of a practical use case, the Traffic Signal Control problem.

Secondly, I introduce MetaPG, a novel domain-agnostic RL design method that prioritizes robustness in addition to performance. MetaPG is an AutoRL method that automates the design of new actor-critic loss functions, represented as computational graphs, for optimizing multiple independent objectives. Through evolutionary search, MetaPG generates Pareto Fronts of new algorithms that maximize and trade all

objectives considered. When applied to a use case aimed at optimizing single-task performance, zero-shot generalizability, and stability on five different environments, evolved algorithms show, on average, a 4.2%, a 13.4%, and a 67% increase in each of these metrics, respectively, compared to the SAC algorithm used as warm-start. Furthermore, MetaPG offers insights into the structure of the evolved algorithms, allowing for a better understanding of their functionality.

Lastly, this thesis focuses on the application of conceptual frameworks and design principles to specific real-world problems in which robustness has been systematically overlooked. I introduce a novel RL system for solving the frequency assignment problem for multibeam satellite constellations. By conducting a comprehensive search over six major design decisions, I identify a design variation that achieves a 99.8% success rate in 100-beam scenarios. However, this variation falls short in handling high-dimensionality and non-stationarity. This thesis demonstrates that robustness against these challenges can be obtained through different design variations, which attain an 87.3% success rate in 2,000-beam cases. Additionally, I also investigate design trade-offs in another real-world application, molecular optimization, and show that current methods are not well aligned with robustness.

Thesis Supervisor: Prof. Edward F. Crawley
Title: Professor, Aeronautics and Astronautics

# Acknowledgments

Firstly, I wish to express my profound gratitude to my advisor, Prof. Ed Crawley, for his guidance throughout my PhD journey. Ed, thank you for taking me on as a visiting student six years ago and later inviting me to stay in your group for graduate school. Your consistent encouragement to pursue my passion and your willingness to help have been instrumental. Many of the pieces of advice you have shared will stay with me throughout my career.

I am also grateful to the other members of my PhD committee, Prof. Cathy Wu and Dr. Aleksandra Faust. Cathy, it has been a privilege working with you, and I greatly appreciate the times you presented me with new learning opportunities and pushed me to go the extra mile. Sandra, the chance to work with you at Google has been a highlight of my PhD journey. I am very grateful for the chance you gave me and for all the generous advice you shared with me. Since "a PhD is not enough", I am taking it to the next step!

I want to profoundly thank my thesis readers, Dr. Yingjie Miao and Dr. Bruce Cameron. Yingjie, your constant availability to share your technical expertise, coupled with your kind-hearted nature, has been a source of strength during these years. Bruce, your unwavering support has been a cornerstone of my PhD journey. Apart from your input and advice on my research, our honest conversations are something I value greatly. Your feedback has allowed me to identify my strengths and areas for improvement, ensuring my graduate school experience was a constant learning journey.

My sincere thanks also go to the various organizations and companies that have supported me throughout my PhD journey. Firstly, I would like to express my gratitude to Fundación Obra Social LaCaixa for awarding me their prestigious fellowship. I am thankful to SES, particularly to Valvanera Moreno and Joel Grotz, for sponsoring my research assistantship at MIT. I have also received valuable technical and computational support from Google. I wish to acknowledge my team: Aleksandra Faust, Esteban Real, Jie Tan, Yingjie Miao, JD Co-Reyes, and Aaron Parisi. Lastly, my gratitude goes to Qurrat Ul Ain and Nadine Schneider from Novartis for their technical guidance.

I feel very lucky for having shared my journey with amazing labmates. To my "PhD siblings", Skylar and Nils, thank you for the shared experiences, constant support, and friendship. I would also like to acknowledge the current members of the System Architecture Group - James, Kir, Ben, and Joel J., and former members who

have been, and continue to be, important mentors: Marc S., Íñigo, and Markus. My appreciation extends to all the undergraduate students who have collaborated with me, especially Sergi and Guillem C. Lastly, sitting in 33-409 has been an amazing experience, thanks to Prof. de Weck, Johannes, Chloé, Michael S., Dan, Elwyn, Rashmi, George, Alex, Anna, Nick, Daniel, and the rest of the members of the Engineering Systems Laboratory. To all of you, I wish you the best in your future endeavors and I hope to stay in touch.

Many people have made my life in Boston and my MIT experience truly unforgettable. To my girlfriend Julia, thanks for your unwavering support, I look forward to the next adventures with you. To my best friend Marc d., sharing the MIT experience with you has made the journey easier in many ways; we both know this is not the end of the ride. To my incredible roommates Manu and Benja, there have been many days in which coming home was the highlight of my day. To my thesis "writing buddies", Ondrej and Maya, thanks for navigating the AeroAstro PhD journey together and for being one of the biggest supports during the last miles. To my other amazing friends: Alex, Armand, Cadence, Carmen, Celina, Dani, David, Faisal, Guillem R., Helena, Inés, Jon, José, Lukas, Regina, Ximo, and everyone at Spain@MIT, thanks for contributing to many moments of joy over the last years, I am grateful for your friendship.

I also want to acknowledge everyone who has supported me from Spain. First, thanks to my parents, Ana and Simó, for always encouraging me to work hard and aim high. Also, thanks to my grandparents, godparents, aunts, uncles, cousins and the rest of my family for motivating me and constantly checking in. I am also deeply grateful to my closest friends in Mallorca, Albert, Mati, Tomeu, Noa, and Vicenç, who always show me that distance is nothing when it comes to our friendship. To my other close friends in the island and in the rest of Spain —you know who you are— thanks for not letting time erode our friendship, I am always there for you.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The purpose of this chapter is to define the main guidelines of this dissertation. It starts with a short introduction on Reinforcement Learning, its evolution, and its growing application to real-world problems (Section **1.1**). Then, I present the main motivation of the dissertation, which focuses on the absence of real-world deployments and the lack of robustness when developing RL for practical applications (Section **1.2**). Next, I introduce the general objectives of this dissertation (Section **1.3**) and extend the initial overview on real-world RL by explaining the different thrusts of work present in the literature (Section **1.4**). Finally, I outline the structure of the remaining parts of the dissertation (Section **1.5**).

## 1.1 Context

Reinforcement Learning (RL) is a learning paradigm based on the sequential interaction between a decision-making entity called the *agent* and its surrounding *environment* [348] (see Figure 1-1). This interaction takes place when the agent observes the *state* of the environment and takes an *action* based on it. This process is sequential, as the environment changes in response to the action, leading to a new state. The goal of the agent is to learn the best action to take in each state, which is encoded in the agent's *policy*. The agent learns by receiving *rewards* after each action, which evaluate the quality of the action in the given state. Using this feedback, the agent improves its policy.

The RL paradigm originated from various research currents that can be traced back to the early $20^{\text{th}}$ century, with studies on trial and error [386] and animal learning [357]. One significant thread was the research on "optimal control" [47], which began

in the 1950s and gave rise to Dynamic Programming and the Bellman equations [34]. This field has continued to evolve in modern times [36]. The study of *discrete* optimal control problems led to the development of Markov Decision Processes (MDPs) [33], which formed the foundation for early RL algorithms like policy iteration [177].

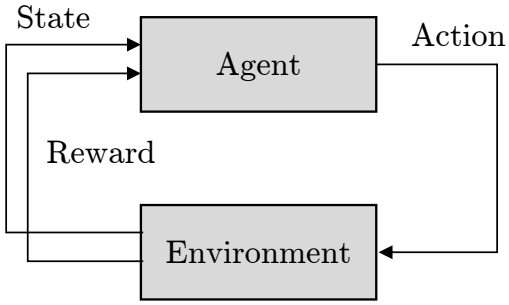



**Figure 1-1:** Representation of the Reinforcement Learning paradigm: an agent interacting with an environment through states, actions, and rewards.

The first concepts of "learning" in RL emerged in the 1970s [385] when trial and error ideas resurfaced after years of focusing on supervised learning for control [215]. Around the same time, "learning automatas" were introduced [363], leading to the research field of Bandit Theory [338]. The formal definition of Reinforcement Learning, as a way to differentiate it from supervised learning, came in the 1980s [25, 27], following the introduction of temporal-difference learning methods [25,26]. Temporal-difference learning research continued from optimal control, and in the late 1980s and 1990s, key methods such as TD($\lambda$) [347] and Q-learning [378] were proposed. This period also saw significant research on RL for board games [349].

Throughout most of the 20$^{\text{th}}$ century, RL employed tabular policies, which mapped states to actions and required registering every possible state in the agent's memory. This constituted a problem when the number of different states was large or when the state was continuous and discretization was not straightforward. The emergence of Deep Learning [144] and its great success in supervised learning tasks such as image classification or natural language processing [327] paved the way for RL using neural networks to replace tabular policies and thus the subfield of Deep Reinforcement Learning (Deep RL or DRL) originated.

While the combination of artificial neural networks and RL was initially proposed as "neurodynamic programming" in 1996 by Bertsekas and Tsitsiklis [38], most advances in DRL research have taken place in the last decade. As shown in Figure 1-2, DRL replaces tabular policies with neural networks, avoiding the need to store all possible states and instead relying on a function approximator. This allows the agent

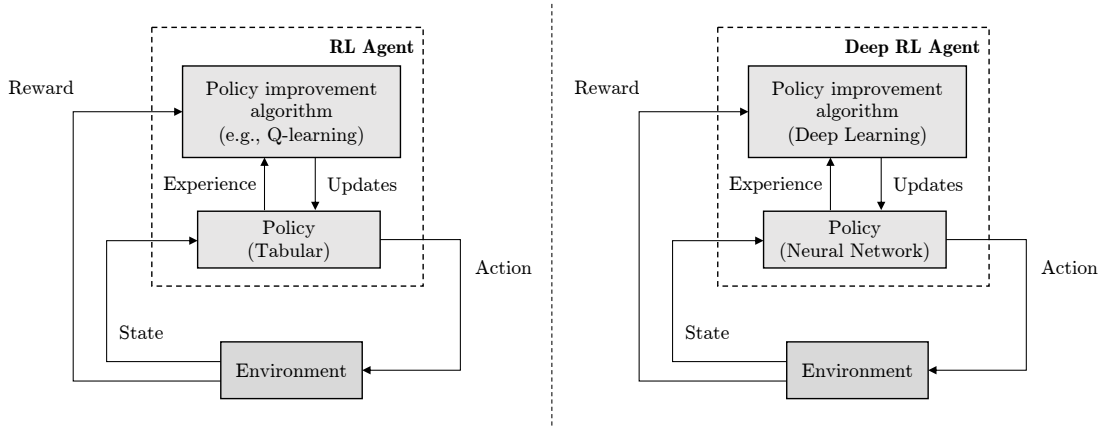



**Figure 1-2:** Comparison of a traditional RL agent (left) with a DRL agent (right), which utilizes neural networks and Deep Learning algorithms for the learning process.

to learn a good policy even if not all states have been visited, as neural networks generalize across the state space. The policy improvement process involves adjusting the network's weights using Deep Learning algorithms, often utilizing objective functions derived from the Bellman equations [34, 270].

In recent years, DRL has been proposed for a wide range of real-world tasks. Various research fields and industries view DRL as a crucial component for future autonomous systems to optimize decision-making (see Figure 1-3). This trend is driven by the evident success of DRL in several key applications [31, 270, 283, 336, 368], and its influence continues to grow. The increasing number of peer-reviewed publications containing the keywords "Reinforcement Learning" or "Deep Reinforcement Learning" in their titles, as shown in Figure 1-4, is a testament to this growth. Moreover, domain-specific studies increasingly emphasize the term *Deep* RL, reflecting neural networks' integral role in advancing the field.

The range of real-world domains in which DRL has been proposed so far is large [230, 274], including —but not limited to— robotics [305], communications [124, 246], drug discovery [104], fluid mechanics [137], autonomous vehicles [208, 351], chip design [269], recommender systems [9], sailing [259], nuclear energy [86], energy systems [303], combinatorial optimization [256], and economics [272]. As Artificial Intelligence (AI) becomes more deeply integrated into society, the potential applications of DRL continue to expand, enhancing human decision-making processes. As Russell & Norvig [324] puts it: "Reinforcement Learning can be viewed as a microcosm for the entire AI problem".

This dissertation studies the application of DRL to real-world problems. A real-

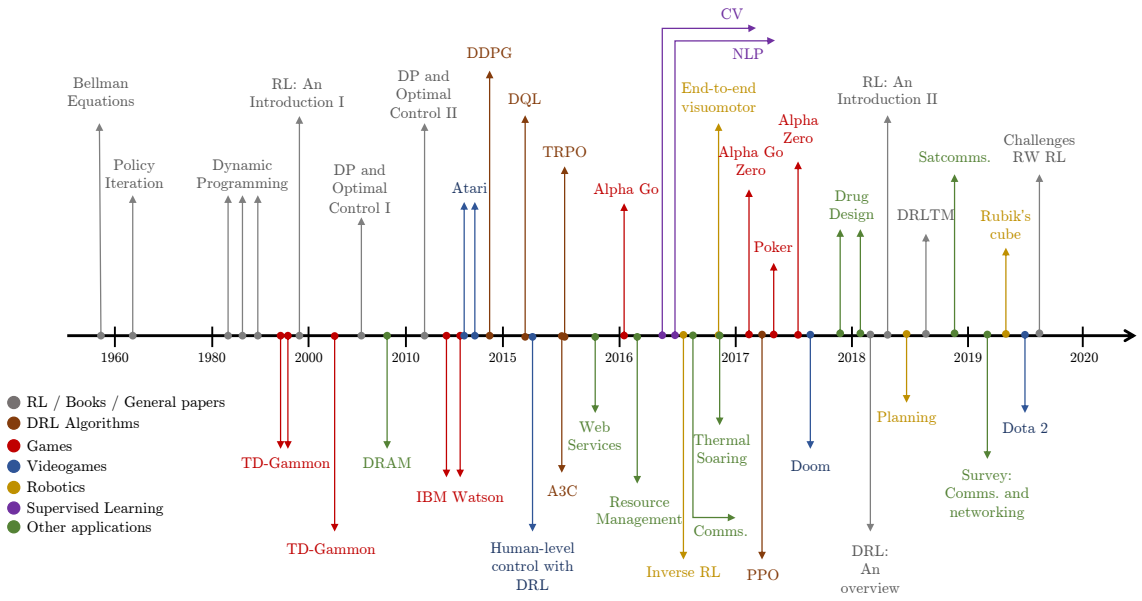



**Figure 1-3:** A timeline of Deep Reinforcement Learning research up to 2020.

world problem is defined as follows

> A **real-world problem or task** is a problem that originates independently from RL or DRL research and has relevance in the physical world. Real-world problems can be fully software-based or may involve hardware components. For example, optimizing resource management in a data center is a real-world problem.

Throughout this dissertation, the terms RL and DRL may be used interchangeably, all referring to Deep Reinforcement Learning, where neural networks and Deep Learning algorithms are integral to the RL agent. However, some of the issues analyzed in this dissertation may also be relevant to traditional RL.

## 1.2 Motivation

In recent years, the market for Machine Learning as a service (MLaaS) has experienced significant growth. In 2017, it was projected to reach US$ 20 billion by 2025 [87]. By 2020, 30% of high-tech and telecommunications companies had already incorporated Deep Learning capabilities into their operations, resulting in an average revenue increase of 5% [258]. However, these estimates fell short as AI technology continued to evolve. The AI Index report of 2023 indicated that the US had invested

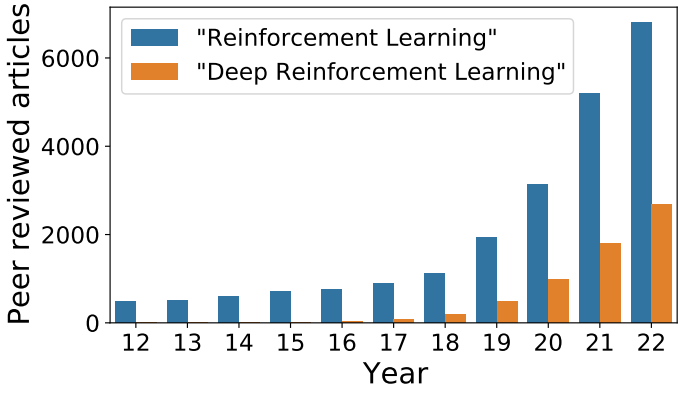



**Figure 1-4:** Number of peer-reviewed articles with the keywords "Reinforcement Learning" and "Deep Reinforcement Learning" in their titles since 2012. Data scraped from Google Scholar.

nearly US$ 50 billion in AI in 2022, and the adoption of AI by companies more than doubled since 2017 [255]. Additionally, AI has extended its reach into other important domains such as policymaking and law. For instance, the US increased its AI spending by 2.5 since 2017, and mentions of AI in global legislative proceedings rose to nearly 6.5 times more since 2016 [255, 261].

One significant factor contributing to the underestimated growth of AI is the progress and accessibility of Large Language Models (LLMs) [121,309]. These models are expected to impact multiple aspects of our lives, from shaping the labor market [103] to facilitating new scientific discoveries [39]. The Reinforcement Learning from human feedback (RLHF) fine-tuning process employed by many LLMs has played a crucial role in aligning these models with human expectations regarding language usage [64, 288]. Prior to the popularity increase of LLMs, RL had already garnered attention in the media and research community due to notable successes such as AlphaGo [336], AlphaStar [368], and more recently, AlphaTensor [111]. Given RL's potential to impact numerous domains, reports emphasize the importance of "charting a course" for RL [260] and "turning toward more general RL" [406] to maximize the value generated by AI.

The reliable integration of RL into autonomous systems operating in the real world not only creates new market opportunities but also addresses significant societal challenges. These challenges include managing the complexities of decarbonizing the energy sector [303], handling capacity and resources in larger, more decentralized, and autonomous communication networks [246], ensuring the safety and effectiveness of intelligent transportation systems [160], accelerating the drug discovery process

[412], solving non-linear and high-dimensional fluid mechanics problems for designing energy-efficient systems [137], and more.

Despite the popularity increase coming from different communities, and the extensive research efforts to prove the applicability of RL in real-world contexts, the successful deployment in real environments is a test many of the proposed RL models in the literature still need to pass —real-world testing is usually left as future work. The number of *reported* deployments remains low, and the actual impact in concrete domains appears to be limited [16, 162, 296]. Currently, significant impact is only observed when there is a substantial allocation of computational and human expert resources [52]. This is particularly concerning in real-world domains in which access to those resources is not easy, preventing many practitioners from fully understanding the applicability of RL in their fields and influencing future research directions. In contrast, other Machine Learning (ML) fields like computer vision or natural language processing offer non-expert practitioners an increasing number of tools to successfully integrate state-of-the-art methods into real-world systems [258]. Consequently, one of the pressing questions for the *extended* RL community is how to make RL technology more accessible and *deployable* without incurring large costs [260].

In this context, a **practitioner** is defined as an individual or entity possessing relevant domain expertise in a specific real-world problem, who may or may not possess expertise in RL at the same time. Practitioners within the extended RL community are those who seek to utilize RL in their respective domains.

The term **deployability**, within the scope of this dissertation, refers to the extent to which an RL agent can be integrated and allowed to autonomously operate in a real-world environment with minimal or no supervision from the practitioner.

### 1.2.1 Why is RL adopted?

As shown in Figure 1-4, the interest in RL and DRL continues to grow. This growth is also evident in the increasing number of fields where RL is being applied to solve specific problems. At a high level, the motivations behind this adoption can be grouped into four categories:

1. **Fast decision-making in high-dimensional contexts:** In certain industries, systems are scaling up and presenting new degrees of freedom, making them

more challenging to operate, particularly in time-sensitive settings. In such cases, DRL offers a new approach to enable fast decision-making. For example, within the satellite communications community, as constellations become larger and more flexible, DRL has been explored as a means to improve decision-making speed [88, 113].

2. **Leveraging raw signals:** Some systems require control policies that can effectively utilize raw signals such as images, sounds, or brain activity. DRL, with its powerful representation capabilities, is being studied to achieve better performance in such cases. Examples of applications include robotics tasks such as manipulation [184, 305] and healthcare applications such as treatment prescription [108, 398].

3. **Supervision is costly or not possible:** In situations where supervised learning is expensive or not feasible, DRL offers a way to learn policies by encoding system-level goals into reward functions and leveraging exploration during training. This is particularly applicable to NP-hard combinatorial optimization problems [96] and drug design applications [281, 306], where DRL can provide approximate solutions and identify promising candidate molecules, respectively.

4. **Long-term dependencies:** Finally, DRL provides a framework that can effectively account for sequential dependencies in decision-making. This aspect is especially relevant for recommender platforms and other interaction-based systems [9, 409].

### 1.2.2 A lack of robustness in real-world RL

As introduced earlier, real-world RL is currently brittle. On one hand, from a domain-specific perspective, it is challenging to fully characterize concrete tasks and environments in the real world, and training in real environments may not always be feasible or preferred. On the other hand, real-world problems, regardless of the domain, involve certain challenges that make learning more difficult. Numerous domain-agnostic challenges have been identified in RL research, including non-stationarity, high-dimensionality, sparse rewards, and generalizability, among others [100, 184, 419]. The combined impact of these challenges affects the ability of an RL agent to successfully complete real-world tasks or problems.

The extent to which an agent accomplishes a real-world task can be examined from two perspectives: performance and robustness. These two properties represent

distinct goals for the agent.

> An RL agent that demonstrates high **performance** in a particular real-world problem or task is capable of satisfactorily completing the task under a specific set of assumptions (e.g., nominal conditions). However, these assumptions may not encompass all the configurations in which the task presents itself in the real world. An RL system that can train a high-performance agent is referred to as a system that achieves high performance.

The majority of domain-specific studies in the literature demonstrate that the proposed RL systems achieve high performance. Under specific assumptions that do not compromise the nature of the underlying problem, it is possible to train an agent that successfully completes the task. Here are some illustrative examples:

- A quadrupedal robot learns to walk on completely flat surfaces using an RL model.
- An RL agent can learn how to efficiently allocate power in a 100-beam satellite.
- New molecules generated via RL show high bioactivity against a certain target.
- A drone can learn to autonomously navigate the interior of large factories using an RL controller.
- An RL agent controls traffic lights at an intersection with 3 roads.

However, achieving high performance might not be enough for an agent to be deployed from the practitioner's perspective, as the agent can only complete the task under a limited set of assumptions that capture specific configurations of the problem. A subsequent step might involve capturing a much broader range of assumptions and configurations, which are necessary for practitioners to consider RL as a more cost-efficient problem-solving approach. This second step requires a different type of ability distinct from performance, which this dissertation refers to as **robustness**. Formally, it is defined as follows:

> **Robustness** refers to the ability of an RL agent to satisfactorily complete a real-world problem or task across all the configurations and sets of assumptions in which the task presents itself in the real world. Thus, agents displaying higher robustness are better equipped to address the specific challenges that arise in real-world RL tasks. An RL system that can train a robust agent is referred to as a system that achieves high robustness.

Robustness, or the lack thereof, primarily determines the *deployability* of RL agents in the real world. To better grasp this concept, let's consider how a lack of robustness manifests in the aforementioned illustrative examples of performance:

- The quadrupedal robot fails to learn gait movement on uneven surfaces, or the trained policy does not generalize beyond flat terrains.
- The RL model for power allocation fails to train effective policies in the 1,000-beam regime.
- RL struggles when faced with more complex targets requiring larger molecules.
- The same drone fails to learn if we impose a budget of samples to learn from.
- The same traffic agent does not perform well in 4-road intersections.

In all these cases, a practitioner may choose not to deploy the agent as it cannot address all the relevant scenarios, and rectifying this issue during operation would likely be more costly than seeking an alternative approach or set of approaches. It should be noted that performance and robustness are not necessarily independent, as an agent that performs poorly is unlikely to be robust. However, the opposite may not hold true.

Given that the literature indicates current RL research prioritizes performance over robustness, a natural question arises: **How can we enhance the robustness of RL systems to real-world problems and phenomena?** This question serves as the motivation for the research direction undertaken in this dissertation. In many cases, addressing robustness aligns with addressing the deployability of RL for the specific problem being considered.

### Other uses of the term robustness

An important distinction regarding the use of the term "robustness" across disciplines is necessary. In certain research areas such as operations research, robustness strictly pertains to the adaptation of systems to uncertainty. In this dissertation, the definition connects robustness to the ability to withstand *change*, specifically indicating that an RL agent is robust to changes in the problem configuration or assumptions that go beyond uncertainty in the parameters. To the best of my knowledge, no other term has been proposed in the RL literature to describe RL's ability to address the complexity of real-world problems comprehensively. Other uses of the term "robustness" in the RL literature include robustness against specific real-world challenges [100] and robustness against uncertainty [393], as derived from operations research.

Robustness also overlaps with the well-known concept of *generalization* or *generalizability* in ML. It might not be uncommon in the literature to refer to an agent that generalizes well as a robust agent. From the ML perspective, a model able to generalize is defined as a model that can complete its task given contexts that have not been seen during training. In the case of RL, an agent that learns a policy that takes good actions given states or scenarios unseen during training is referred to as an agent that generalizes or possesses generalizability. For example, an RL agent trained to drive in cities should generalize to unseen traffic conditions, as it cannot be trained in all possible scenarios. This issue, which in Chapter 6 is referred to as zero-shot generalizability, is one of the real-world challenges considered by many [100,184,210,393].

However, generalization can also be seen from the perspective of the complete RL system. To better understand this, we can think of real-world problems as a composition of *subproblems*, which correspond to different sets of assumptions or different configurations. For example, the problem of driving could be decomposed into two subproblems: driving in a city and driving on a highway. Similarly, modifying elements of the learning process such as the training budget or restricting access to specific information from the environment, can also be regarded as considering different subproblems. In that sense, one agent can be trained to perform well in one specific subproblem, although it can only be considered *robust* when it is able to *generalize* across all important subproblems of the real-world problem at hand. This idea of generalization and its overlap with robustness is further discussed in Chapter 3. The subproblem perspective is introduced in this dissertation to support the explanation of certain concepts; in practice, partitioning a problem into subproblems might not be straightforward, might have multiple levels of decomposition, and might be subject to different interpretations.

## 1.3 General Thesis Objectives

This dissertation sits at the intersection of RL research and real-world applicability, aiming to bridge the gap between RL system design and adaptation to real-world requirements. A significant aspect of this gap is the lack of robustness in current RL agents, which serves as the overarching framework for the dissertation's general objectives. Conversely, improving performance and exploring ways in which RL beats performance benchmarks for specific practical applications are beyond the scope of this work.

Given that the holistic perspective of robustness is often neglected in the RL

literature, the first objective of this dissertation is to gain clarity on this concept and identify its various manifestations in the real world. The specific aim is as follows:

**General Objective 1**

To characterize different aspects of robustness that hinder the deployment of RL systems and agents in real-world scenarios.

Then, I aim to better understand what needs to change in current RL design practices if robustness and the bridge with real-world applicability need to be prioritized. In this regard, I aim to investigate new methods and practices that align better with robustness. While increasing performance is not the primary focus of this dissertation, I will consider it simultaneously and identify directions that promote both performance and robustness, as well as directions that inherently involve trade-offs between the two. The second objective of this dissertation is stated as follows:

**General Objective 2**

To develop new RL design methods that prioritize robustness alongside performance and comprehend their associated trade-offs.

Lastly, while the conceptual analysis in this dissertation and many challenges related to real-world RL and robustness are domain-agnostic, I acknowledge that implementing robust RL systems may require considering the specifics of individual real-world problems. To this end, I will focus on particular real-world problems where robustness can be measured and apply the ideas and methods explored in this dissertation to obtain more robust solutions for those specific problems. The final objective of this dissertation is defined as follows:

**General Objective 3**

To apply conceptual frameworks and design principles that prioritize robustness to address concrete real-world problems.

The next chapter will review the existing literature on real-world RL and its associated challenges to gain a deeper understanding of the research gap. Subsequently, specific research opportunities that this dissertation aims to address will be proposed, along with a formal Thesis statement.

## 1.4 Background

The majority of research around real-world RL can be classified either as domain-specific or domain-agnostic, two complementary thrusts of work. In this section I introduce both of them and present representative examples. In addition, I also discuss other areas of RL research that are not directly linked to real-world RL — although they might impact it— such as the study of theoretical foundations of RL, the development of new RL algorithms, or the application of new Deep Learning research to RL. I consider these as research thrusts that aim to enrich the toolbox available to researchers who do focus on specific issues of real-world RL.

### 1.4.1 Domain-specific real-world Reinforcement Learning research

Domain-specific RL research focuses on the application of RL to a specific real-world problem or task with the objective of solving that task or improving upon the state-of-the-art. The majority of the works come from specific communities; examples include using RL for Traffic Signal Control [278] or RL for drug discovery [81]. In each of these examples, and in many others from numerous research communities, RL studies mostly focus on finding existing RL methods that can work well for the problem being considered.

To evaluate specific agents, designers generally pick a *subproblem* (see Section 1.2.2) or a few of them, and run analysis using the experimental setup defined by these subproblems. While the subproblems being considered might be good representatives of real-world scenarios, most experimental setups do not capture well subproblem diversity [192]. This leads domain-specific research to overfocus on performance in those subproblems rather than designing for robustness in the set of —possibly many— subproblems that matter for deployability.

In the process, designers commonly leverage broad domain expertise to finetune the different components of RL systems, such as state representations or reward functions. However, in many cases, these efforts lead to excessively tailoring agents to the few subproblems that conform the experimental setup. To try increasing the variability in the experimental setups, some communities put substantial efforts in designing domain-specific RL benchmarks, such as OpenSpiel in the case of games [224], and in developing flexible simulators that can be incorporated in experimental setups, such as the SUMO library in Traffic Signal Control [241].

Table 1.1 summarizes the main characteristics of domain-specific RL research and contrasts them with the trends of domain-agnostic RL research.

### 1.4.2 Domain-agnostic real-world Reinforcement Learning research

Domain-agnostic RL research consists of developing algorithms, methods, and frameworks that address specific problem features that manifest across various domains. For instance, coming up with new approaches that benefit robustness towards a specific real-world challenge falls into the class of domain-agnostic RL research. The ultimate goal of these investigations is that their findings can be applied to multiple domains and problem instances.

There are many studies in the literature in which the main premise is to pick a specific real-world challenge and investigating it without using any concrete real-world problem as support. Generally, the contributions of these works involve the design of a new method or approach that moves the community a step closer to achieving robustness against that challenge. To that end, these studies utilize standardized benchmarks that are not based on specific real-world problems but on toy-like experimental setups which emphasize the presence of the challenge being addressed. The research on these benchmark environments is rich [28, 75, 84, 100, 147, 286, 355, 361, 373, 400].

While one does generally observe improvement upon the benchmarks for the challenges considered, rarely do these studies follow through with real-world testing on specific applications. Therefore, evaluation on real-world experimental setups is usually left as future work for domain-specific practitioners. A possible reason for that might be to avoid tailoring the new methods to the features of any domain in particular, which might not generalize to other domains. In that sense, domain-agnostic RL research rarely leverages domain knowledge, many important domain skills are left to be learned from scratch by the agent.

Table 1.1 summarizes the points presented in this section and juxtaposes them with their domain-specific counterparts.

### 1.4.3 What is missing?

After presenting a description of the trends in both domain-specific and domain-agnostic, here I argue about the possible synergies that could be established between both thrusts of work in order to make steps towards more robust RL and a larger

**Table 1.1:** Main differences between domain-agnostic and domain-specific real-world RL research.

| Domain-agnostic research | Domain-specific research |
|---|---|
| Focuses on challenges of real-world RL without investigating any particular problem or task | Focuses on real-world problems and tasks, although it generally does not capture all present real-world challenges well; it focuses on single subproblems |
| Investigates new methods that address the specific challenge being considered | Investigates which existing methods are useful to address the problem being considered |
| Studies robustness against the real-world challenge at the center of the research | Generally prioritizes performance in one concrete subproblem over robustness across many subproblems |
| Experimental setups mostly include toy-like environments and domain-agnostic RL benchmarks that have substantial flexibility | Experimental setups are based, totally or partially, on real-world environments, although they mostly capture few subproblems |
| There is a widespread lack of follow-through on specific real-world problems that exhibit the challenge being considered | Leverages domain expertise to test on real-world-based environments |
| Sometimes attempts to learn certain behaviors and constraints from scratch | RL systems and agents are tailored to a specific subproblem |

number of deployments. I summarize my perspective on these opportunities in the following bullet points:

- The path to robustness and deployments passes by doing research on specific real-world problems while focusing on both the scope of important subproblems and the real-world challenges that are intrinsic to the problem.

- In some cases, existing methods might be enough to address the problem, and a know-how on leveraging those is an important skill. In other cases, the solution might require investigating new methods and designs.

- The key to deployability is to address robustness capturing all intrinsic real-world challenges and considering all important subproblems. In the process, thinking from the point of view of the robustness to individual challenges could be helpful. To that end, practitioners might need to trade performance in concrete subproblems for robustness in the overall problem.

- Experimental setups should be based on the real-world problem at hand, although they should also be flexible enough to model subproblem variety.
- There should be a leverage of domain expertise without getting into excessive tailored solutions that lack robustness.

### 1.4.4 Other areas of Reinforcement Learning research

In addition to domain-specific and domain-agnostic RL research, there are several other research areas that are out of the scope of this dissertation but contribute to the progress of RL as a field and directly or indirectly influence real-world RL research. Here I refer to RL theory, the development of new RL algorithms, the infusion of Deep Learning research into RL, and the development of new research infrastructure for RL. In the following lines I briefly summarize the main ideas of each research direction.

Theoretical foundations of RL delve into the mathematical principles underlying RL algorithms. Researchers analyze the convergence properties of different RL algorithms, investigate their sample complexity, and develop theoretical frameworks to understand their behavior. By establishing theoretical guarantees, researchers can better guide the development of RL algorithms, ensure their stability, and provide insights into their limitations and potential improvements. A good example of theoretical research in RL is the study of multi-armed bandits [338].

Developing new RL algorithms constitutes an active area of research which aims to develop new methods for RL agents to learn more effective policies. By identifying problems in the learning behavior of agents, researchers propose new algorithms that can be applied in different domain-specific contexts. In that sense, these algorithms belong to the "toolbox" that domain-specific practitioners use when applying RL in their domains. In many cases, research directions on new RL algorithms overlap with the theoretical study of RL and also synergize with Deep Learning research. Examples in this category include many popular RL algorithms such as Rainbow [166], Proximal Policy Optimization (PPO) [328], and Soft Actor-Critic [152].

Then, the application of new Deep Learning research to RL leverages the advances in neural network architectures, optimization methods, and training techniques. Using several techniques first proposed for supervised learning research has been successful in many RL studies and constitutes an important source of innovation in RL. The literature in RL already shows examples leveraging graph neural networks (GNNs) [58], transformers [318], automated Machine Learning (AutoML) [296], trans-

fer learning [422], etc. These methods also become part of the design toolbox that RL researchers and practitioners can use.

Finally, there are also efforts focused on developing RL infrastructure that other research threads can rely on. The most well-known examples correspond to the creation of widely-used repositories for RL development such as OpenAI Gym [43], OpenAI Spinning Up [6], and DeepMind ACME [171]. In addition, other authors directly work on highly-flexible environment suites that can be leveraged for multiple purposes, such as the Arcade Learning Environment [30] and the StarCraft environments [326].

## 1.5 Thesis structure

The remainder of this dissertation is structured as follows: Chapter 2 reviews the literature of different domain-agnostic fields that are relevant for real-world RL, including the real-world challenges and a deeper dive into AutoML for RL. Chapter 3 introduces a roadmap for real-world RL that aims to identify all the important elements that play a role in the performance and robustness of RL agents in practical applications. Chapter 4 dives deeper into the current real-world considerations for one of these elements, the state space, and studies the design of feature sets based on mutual information in the context of policy learning. Then, Chapter 5 proposes a new AutoML method, MetaPG, for the design of RL loss functions that optimize multiple objectives at the same time. Chapter 6 demonstrates the validity of MetaPG by applying it to the specific use case of optimizing, in addition to single-task performance, zero-shot generalizability and stability, two key aspects of RL robustness. Chapter 7 shifts the focus to the implications of developing new RL systems in the context of concrete real-world problems. In Chapter 8 and Chapter 9, I focus on two particular problems, frequency plan design in satellite communications and molecular optimization, respectively, and propose new RL systems and evaluation procedures that better address aspects of robustness for these particular problems, analyzing the specifics of each problem in the process. Finally, Chapter 10 summarizes the work conducted in this dissertation, outlines the main contributions, and identifies areas of future research.

# Chapter 2

# Literature Review

This chapter covers the literature review related to the main goals of the dissertation. Section **2.1** explores the different issues that make real-world RL challenging and ties them to different areas of the literature. Section **2.2** presents the literature on the so-called challenges of real-world RL and summarizes the different approaches utilized to mitigate them. Then, it goes deeper into one of the most recently popular areas of work: Automated Machine Learning for RL. Following, Section **2.3** presents other important considerations for real-world RL research such as reproducibility and experimental analysis. Next, Section **2.4** provides different examples in which RL has been successfully deployed in the real world. After examining the general literature on real-world RL, Section **2.5** presents the specific research opportunities addressed in this dissertation and Section **2.6** outlines the research statement. The chapter concludes in Section **2.7** with an overview of which opportunities are addressed in each of the chapters to follow.

## 2.1 What does make the real world hard?

As discussed in the previous chapter, one of the robustness issues that impede RL agents from succeeding in the real world is the presence of domain-agnostic *challenges* that manifest across multiple domains. Initially, the majority of these challenges were not directly incorporated into the theoretical RL formulations and have only been recognized as RL research and application have both progressed. While there is no well-established classification for these challenges, several studies have attempted to summarize them, I plot them in Figure 2-1.

Dulac-Arnold et al. conducted extensive work on these challenges in their pa-

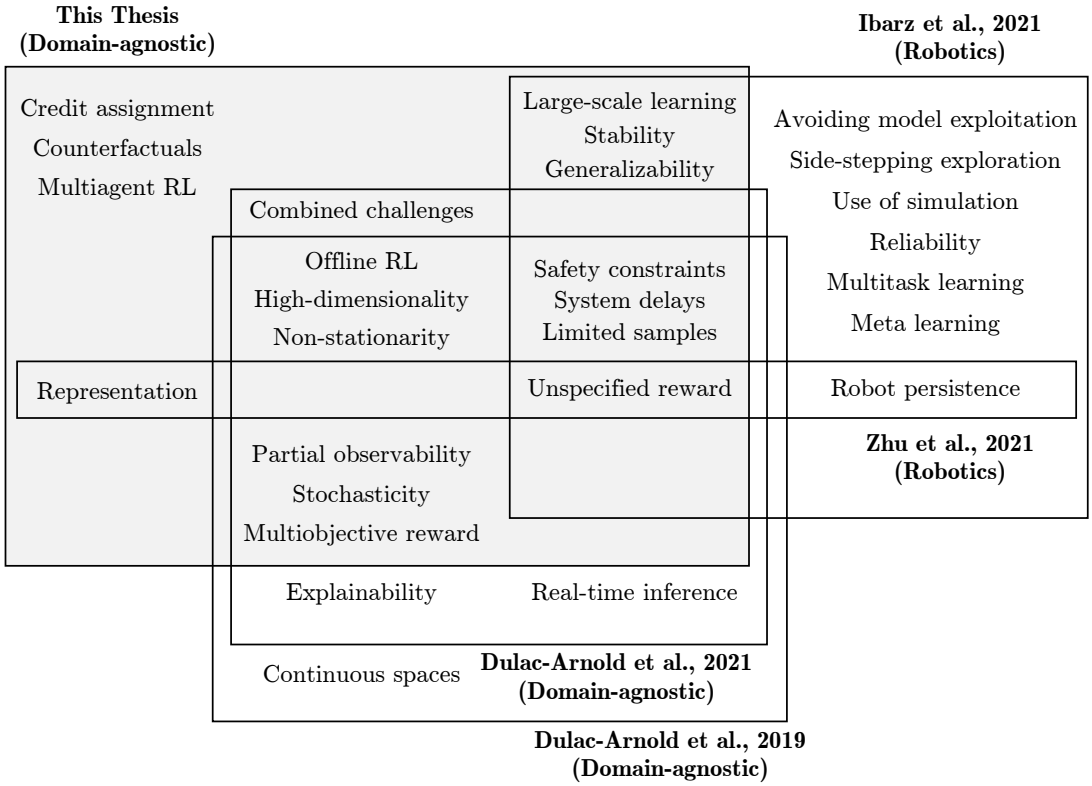



**Figure 2-1:** Set of challenges of real-world DRL considered in different studies and in this dissertation. Some of the challenges might interact or overlap in specific settings.

pers [102] and [100]. They offered a comprehensive review of nine domain-agnostic challenges and analyzed their impact on RL agents. These challenges include learning from limited samples, dealing with delays in the system, high-dimensional state and action spaces, safety constraints that hinder exploration, partially observable environments, multi-objective or unspecified rewards, real-time inference, training from offline logs, and providing explainable policies. By introducing a set of real-world-oriented benchmarks in [100], the authors were able to quantify the correlation between the intensity of a specific challenge and the decrease in performance when using two state-of-the-art DRL algorithms, namely D4PG [24] and DMPO [3]. This study stands as the first holistic analysis of the robustness of RL in real-world settings. One of the key findings from their work is that the simultaneous presence of multiple challenges significantly impacts performance. Since that many real-world problems involve multiple challenges concurrently, this paper sheds light on why operators might hesitate to deploy even the most advanced RL models.

Other authors have approached the challenges of real-world RL from the perspective of specific domains, particularly robotics. Zhu et al. [420] emphasize three

important challenges for RL in robotics: learning from raw sensory input, devising reward functions without reward engineering, and learning without resets. The authors propose a method that specifically addresses this triad while recognizing that achieving "truly scalable real-world robotic learning" involves tackling a broader range of challenges simultaneously.

This broader perspective is adopted by Ibarz et al. [184] as they outline twelve different DRL challenges specific to robotics: reliable and stable learning, sample efficiency, use of simulation, side-stepping exploration challenges, generalization, avoiding model exploitation, robot operation at scale, asynchronous control, goal-setting and reward specification, multitask learning and meta learning, safe learning, and robot persistence. For each challenge, the authors discuss specific manifestations of each challenge in robotic tasks and review the related literature. Their conclusions suggest that, while there exists a set of real-world DRL challenges that are truly domain-agnostic, specific domains might introduce unique additional challenges that are equally important.

The aforementioned papers aim to analyze the various challenges posed by real-world operation to DRL models. Their findings can be summarized as follows:

1. The set of domain-agnostic challenges is broad and diverse, and there is not a unique interpretation of this set. Capturing all possible challenges requires an holistic perspective on RL operations.
2. The nature of these challenges may vary depending on the specific application. Certain fields, such as robotics, may prioritize a subset of challenges or exhibit additional domain-specific challenges that are equally crucial.
3. In general, real-world RL challenges often manifest in *combinations*, and the cumulative effect can significantly hinder the performance of RL agents.

To the best of my knowledge, no other paper in the literature has thoroughly examined multiple real-world RL challenges or analyzed RL robustness from a holistic standpoint. Most existing literature tends to focus on specific challenges or issues rather than considering the full scope of robustness. The subsequent section provides a literature review on each of the specific challenges considered in this dissertation.

## 2.2 Challenge mitigation strategies

This section addresses the literature on each specific challenge and tries to summarize the mitigation strategies adopted for each of them. These are not unique; sometimes

the same method is proposed in different contexts. The goal of this section is to extend the challenges identified in [101] and try to carry out an analysis following the format in [184]. This part of the dissertation is based on my subjective view of the challenges and the reviewed literature; as an active area of research, it remains open to different interpretations.

As shown in Figure 2-1, this dissertation considers 18 challenges: offline RL, learning from limited samples, large-scale learning, high-dimensionality, safe RL, partial observability, non-stationarity, unspecified reward, multi-objective reward, system delays, representation, generalizability, long trajectories and credit assignment, stochasticity, multi-agent DRL, counterfactual reasoning, stability, and the combination of many of these challenges.

Note these challenges do not need to be independent, in some contexts specific challenges might be a consequence of other challenges manifesting (e.g., non-stationarity during learning might be a cause of a partially observable environment). There are two other challenges highlighted by some authors that are not included in my analysis as isolated challenges: continuous spaces and real-time inference. These challenges are hard to be found being addressed in isolation and therefore are excluded from the previous list. However, they remain important considerations for operation; the former deals with how state and action spaces that are naturally continuous can be successfully incorporated into the agent's representations, and the latter concerns the speed in which an agent takes an action in the environment after receiving the updated states. In some cases, a high frequency might be needed, this problem is linked to hardware architectures and has been studied by the deep learning community [197, 271].

Before diving into the challenge-specific review, it is important to mention that the literature on each challenge is in itself rich enough. However, my goal is to provide the reader with a sufficiently deep review of each challenge such that, when considering the overall set of challenges, the reader can comprehend the various approaches employed, their classification, and the most prominent ones. To facilitate further consultation, I also provide review papers for most of the challenges that delve deeper into the RL community's work on each specific challenge.

### 2.2.1 Challenge-specific work

**Offline DRL** Some agents might require learning from offline logs or external policies instead of directly interacting with the environment, as that might be costly or

not possible. An extensive review on the subject is presented in [228]. To address this issue, different off-policy algorithms such as DDPG [233] or D4PG [24] can be used in some cases. Other authors propose batch-constrained RL methods [128,222,334,388], where the learned policy is constrained based on the state-action distribution from the dataset and the extrapolation error is accounted for. Then, the work by Agarwal et al. [12] considers an ensemble or convex combination of Q-functions to leverage data in a replay buffer. Finally, model-based RL [348] constituted another research area in the context of offline DRL [205, 401].

**Learning from limited samples** In some cases an agent must learn from a small number of training samples, either because acquiring experience is slow or costly, or rapid adaptation to a new context is needed. While the representations chosen or learned, as well as the specific algorithm —e.g., SAC [153]— can impact the learning speed [340], multiple methods have been proposed to directly address data efficiency. One alternative is to learn a model of the world and use that model to plan [48,66]. In the context of learning specific tasks, if expert demonstrations [98, 285] or behavioral priors [337] are available, the agent can bootstrap from those to avoid interacting with the environment. If the goal of the agent is multitask learning, various tasks can be learned concurrently taking into account multiple gradient inputs [399], or if the tasks are to be learned sequentially, meta learning algorithms, especially few-shot methods, offer a way to learn new tasks faster [115, 227, 231, 346]. Finally, in online learning contexts, where new tasks need to be learned fast and on-the-fly, different approaches have been proposed to promote forward and backward transfer [53, 248, 275, 329] and avoid catastrophic forgetting [211].

**Large-scale learning** In specific settings an agent should be able to quickly capitalize on massive amounts of data, either because experience comes at a high frequency or multiple independent agents can collect experience simultaneously. For the latter case, when environments can be parallelized (e.g., self-driving cars, recommendation systems, drone swarms), distributed training with importance and priority mechanisms has been proposed in different works [8, 107, 172].

**High-dimensionality** Some agents might need to operate in high-dimensional or combinatorial state and/or action spaces (e.g., natural language, molecular space, online retail catalogs). Here, one approach is to operate with lower dimensional embeddings of the spaces [99, 322]. Zahavy et al. propose action elimination mech-

anisms to determine which actions not to take first [402]. For the specific case of Q-learning [348] over large action spaces, de Wiele et al. [83] propose replacing the maximization operator for a neural network. Finally, the use of canonical spaces can help reducing the state space size by encapsulating redundant states together [387].

**Safe DRL** While exploration plays an important role in the success of RL, autonomous agents operating in real-world environments should account for safety constraints and be able to evaluate risks. One common approach to that end is to encode constraints as part of the reward function [136], but that might not always be desirable [7]. Some studies propose adding a learnable safety layer on top of the policy in order to prune or correct unsafe actions [80]. The work by Tessler et al. [356] explores reward shaping and proposes a method that subtracts constraint-violation penalties to the reward. Then, Lyapunov functions have also been proposed to certify stability and safety of different RL-based controllers [35, 63]. Constraint satisfaction can also be guaranteed by means of primal-dual methods, as shown by Qin et al. [308]. Finally, an agent can also learn to trade rewards and costs by specifying constraints as costs with state-dependent and learnable Lagrangian coefficients [40].

**Partial observability** Many environments in the real world are partially observable. In the context of DRL, some authors initially proposed incorporating past observations to the state [270] or use recurrent neural network architectures [159]. Inspired by the theory on POMDPs [50], Igl et al. [187] propose training a variational autoencoder to learn latent representations encoding belief states. In the case the agent competes against other non-fully-observable agents, Jaderberg et al. [190] show that training populations of agents eventually leads to best agents finding suitable policies for the environment. If, instead, the agent must cooperate with other agents, the use of shared experience replay helps mitigating the effect of partial observability [282].

**Non-stationarity** A robust policy should be effective in non-stationary environments, where the underlying dynamics might change over time due to various factors that introduce noise or perturbations. In these contexts, one alternative is the use of latent variables that encode environment representations that are robust to noisy cues [391]. A well-established approach is to assume uncertainty in the transition matrices and derive robust algorithms that consider worst-case scenarios [250] or pursue soft-robustness [92]. Bayesian optimization-based methods can be also derived from

this latter idea [91]. Finally, data augmentation and randomization during training can also lead to policies that adapt to real-world environments and generalize better [301].

**Unspecified reward** Sometimes agents must learn skills without clear reward signals, due to either unavailable expert feedback, complex exploration dynamics, or long horizon tasks. If there is no reward function but expert demonstrations are available, Inverse RL is an approach to learn reward signals [127]. In some scenarios Inverse RL is preferred over imitation learning, a more direct and scalable approach [117, 118]. Hansen et al. [156] propose a method to train policies by means of self-supervised learning when deploying in environments without reward information. Another alternative is to learn a goal-conditioned policy via unsupervised learning, maximizing the similarity between the visited states and a goal state [377]. In the context of multitask learning, Eysenbach et al. [109] show an agent is capable of learning a diverse set of distinguishable skills by maximizing entropy. These skills can be then used to better adapt to new tasks.

**Multi-objective reward** Several tasks in the real world require accounting for multiple objectives and an agent must learn to reason about them and make trade-offs if necessary. To that end, many works rely on scalarization approaches that combine the different objectives into a weighted reward function. This approach can be hard to tune if there are changes on the individual rewards' scale or their priorities over time. To have a better control over the objectives, [2] proposes training individual policies for each objective and then, instead of combining rewards in the reward space, combine policies in the distribution space. Another alternative is to train a different policy per preference over objectives [392, 396], which leads to dense Pareto-optimal sets of policies that trade the different objectives following the operator's preferences. Finally, meta learning methods have also been proposed to learn new objective preferences and automate reward search in a few-shot fashion [59, 110].

**System delays** DRL experimental setups generally assume negligible delay when operating, observing new states, or receiving rewards. That might not be the case in the real world. To address this issue, the framework of Delay-Aware MDPs was introduced by Chen et al. [54] to account for delayed dynamics. A similar idea was proposed by Derman et al. [90], where the delayed-Q algorithm leverages a forward dynamics model to predict delayed states. Other works propose training with arti-

ficial delays to simulate the hardware gap [221]. In the context of recommendation systems, the method by Mann et al. [252] exploits intermediate observations/symbols to mitigate the effect of delays.

**Representation** In certain environments the challenge lies in efficiently encoding all information relevant to the problem or task, leveraging the sufficient amount of domain knowledge or inductive biases [167]. Trade-offs are present, e.g., learning policies from physical state-based features might be more sample-efficient —although not always possible— compared to learning from pixels or other raw signals [355]. The question "what makes a good representation for RL?" is studied by Singh et al. [337]. A simple approach is to design different representations for the same environment and turn the specific chosen representation into a hyperparameter that can be tuned based on the scenario [132, 207]. Different environment encodings can be also combined into multimodal representations (e.g., image and sound in video-based environments) [358, 362]. Helpful representations can also be learned, for instance by means of contrastive learning frameworks [340, 389]. Then, Zhang et al. [403] propose learning invariant representations by means of lossy autoencoders that capture only task-relevant elements. Finally, representation problems can also be regarded from the perspective of the reward; better reward functions might be devised following reward shaping methods [60, 110].

**Generalizability** Also referred to as generalization in the literature. Policies should be able to generalize to different instances of the system/and or environment regardless of their low-level features, without posing a considerable challenge. To that end, randomization strategies can be used to increase robustness against poor generalization [14, 226, 302, 359]. AutoML methods for RL serve as another way to achieve generalization, by parametrizing specific elements of the DRL framework and using an outer loop learner trained on multiple environments [15, 176, 212, 280]. Learning common invariant latent spaces could be another approach to consider in some contexts [148]. The use of regularization strategies has also proved to benefit generalization [56, 69, 186].

**Long trajectories and credit assignment** When trajectories are long and/or rewards are sparse, learning effective behaviors can be challenging; the agent must discover a long sequence of correct actions. Hierarchical RL poses a possible solution, by considering a hierarchy of auxiliary tasks with known reward structure in order

for the agent to reason at different levels of temporal resolution [273, 319, 365]. Other works propose attention mechanisms to ease credit assignment over long timescales [182, 379]. Then, the method presented by Arjona et al. [17] tackles the problem by redistributing reward, i.e., creating a return-equivalent MDP that redistributes reward more uniformly. Finally, reward shaping methods are also studied for this type of contexts [60, 344].

**Stochasticity** In some occasions, real-world environments can be stochastic, which might lead to high variance gradient estimates that hamper learning. To make sure the agent is trained over a wide distribution of states, data augmentation strategies and randomization are proposed by some authors [226, 359]. The method presented by Ghosh et al. [141] suggests partitioning the initial state distribution and train different policies that are later merged in a divide-and-conquer fashion. In highly-stochastic environments, the final policy might be better if the agent does not learn based on the average return but on a distribution over returns [24, 32, 79].

**Multi-agent DRL** In many real-world environments (e.g., autonomous vehicles, robot swarms), a team of agents must align their behaviors while acting in a decentralized way [312]; leveraging experience from multiple agents is not always straightforward and other challenges such as partial observability and non-stationarity might also come into play. An extended review on the subject can be found in the work by Nguyen et al. [277]. To address this challenge, one approach is to have each agent learn independently [352], which decentralizes training but might originate stability problems [123]. On the opposite side, Foerster et al. [122] explore the framework of multiple decentralized actors and a single centralized critic. Inspired by Value Decomposition Networks [345], the work by Rashid et al. [312] and Son et al. [339] propose hybrid mechanisms to combine per-agent Q-functions into a single centralized Q-function. Multi-agent Policy Gradient algorithms introduce a similar concept designed for continuous spaces [229, 243], which can be also combined with attention [188].

**Counterfactual reasoning** The ability to reason about not taken actions and "what-ifs" is necessary in some real-world systems, especially when constraints or risks are hard to capture. This is a relevant problem in healthcare applications [307]. This challenge in part overlaps with offline DRL, since extrapolation techniques can be useful in some contexts, especially when there is correlation between state-action

pairs inside and outside the experience databases [128]. While some studies might touch on this concrete challenge, I did not find any work specifically focused on counterfactuals and real-world DRL. Facebook's platform Horizon [139] leverages work on Counterfactual Policy Evaluation [376] to evaluate policies without needing to deploy them. In the specific case of agents playing imperfect-information games might benefit from Counterfactual Regret Minimization strategies [46].

**Stability** Once deployed, agents should maintain the desired behavior for indefinite time, even when new experience is collected. This challenge is connected to other considerations: online learning and the problem of catastrophic forgetting, autonomous resets (Ibarz et al. [184] identify autonomous resets as one of the specific challenges in the context of robotics), and the general issue of reliability. From the perspective of post-deployment or operation, there is no literature that addressed this issue in the context of real-world RL. However, in some control applications stability has been studied as the ability to reach the same performance during independent training runs [21]. This is an issue that overlaps with stochasticity, as it has been shown that randomness can play a substantial role in the outcome of a training run [163].

**Combined challenges** Finally, as pointed out by Dulac-Arnold et al. [101], real-world DRL challenges do not usually appear in isolation but combined. The literature specifically addressing multiple challenges simultaneously is scarce. For instance, the work by Jaderberg et al. [190] focuses on both multi-agent settings and partial observability, although they are commonly related. There are no other works tackling numerous challenges at the same time.

### 2.2.2 Summarizing the mitigation approaches in the literature

Subject to the previous analysis, one can realize that, while the set of challenges is diverse, the mitigation strategies used to address them are not unique to one challenge but sometimes the same method or approach is proposed in different contexts. Table 2.1 tries to group the different different strategies into 13 classes or types: AutoML frameworks, derivation of mathematical theorems, changes in neural network architectures, solutions directly derived from RL theory, embeddings and latent spaces, reward engineering, derivation of environment models, pruning and masking strategies, relying on auxiliary tasks, data augmentation approaches, use of heuristics, population-based methods, and solutions specifically designed for multi-agent cases.

**Table 2.1:** Summary of the different approaches that have been considered by the RL community to address the specific challenges of real-world RL. Each approach has been considered for different challenges independently.

| Approach | Description | Examples |
|---|---|---|
| AutoML | An outer loop learner changes meta parameters to better adapt to the challenge | Meta learning for offline DRL, meta learning for multi-objective reward, Meta RL for generalizability, replacing action maximization by neural network search in high-dimensional spaces |
| Mathematical guarantees | Derive equations and theorems that support the challenge fulfillment | Lyapunov functions and primal-dual methods for safe DRL, assume uncertainty matrices to address non-stationarity |
| Neural network architectures | Rely on Deep Learning advances to increase robustness against the challenge | RNNs for partial observability, attention mechanisms for credit assignment, network ensembles for offline DRL |
| RL theory | Adapt theoretical RL frameworks to DRL settings and the use of neural networks | Off-policy algorithms, POMDPs, Delay-Aware MDPs, Constrained MDPs, Maximum-entropy RL |
| Embeddings and latent spaces | Address challenge problems by relying on robust intermediate embeddings | High- to low-dimensional embeddings, latent variables for non-stationarity, multimodal and contrastive learning-based representations, unsupervised learning |
| Reward estimation or modification | Try to overcome the challenge by directly modifying the reward structure and/or function | Reward shaping in long trajectories, reward shaping for safe DRL, reward redistribution, distributional RL against stochasticity |
| Deriving a model | Instead of learning a policy, learn models of the environment and use them to plan | Model-based RL, imitation learning, inverse RL |
| Pruning and masking | The learning process involves deciding, among different learning signals, how important each of them is and eliminating the unnecessary ones | Batch-constrained methods for offline DRL, distributed training in large-scale settings, action elimination in high-dimensional spaces, divide-and-conquer methods in stochastic environments |
| Use auxiliary tasks | Provide the agent with auxiliary tasks that, combined, increase robustness against the challenge | Multitask learning and online learning for data efficiency, self-supervised learning for unspecified reward, hierarchical RL in long trajectories |
| Data augmentation | Rely on different data augmentation and data wrangling techniques | Randomization to transfer better, data augmentation to address non-stationarity and stochasticity |
| Heuristics | Use human-crafted rules or processes to address the challenge | Scalarization of multiple objectives, hyperparameter tuning |
| Population-based methods | Have multiple agents with slightly different parameters/objectives and search for the best ones | Multi-agent populations in partially observable environments, multi-objective populations |
| Multi-agent specific | Solutions specific to the multi-agent challenge that can not be mapped to other challenges | Independent Q-learning, decentraliced actors and centralized critic, hybrid mechanisms |

The number of ways in which real-world challenges, which constitute different aspects of robustness, are addressed is large, but there is not a clear one-to-one mapping between specific strategies and specific challenges, since the same ideas can be applied to solve different problems. For example, deriving a model of the environment is useful for tackling offline RL problems and also cases in which the reward is unspecified. Despite the diversity of solutions, I identify a trend in automated Machine Learning (or AutoML) frameworks to solve some of the challenges. In the following section, I

expand my literature review on AutoML and identify current research threads.

### 2.2.3 AutoML for RL: a novel and popular research direction

Based on the grouping in Table 2.1, many research directions are converging towards automated Machine Learning or AutoML [183] as a promising research area to tackle domain-agnostic challenges. AutoML has proven to be a successful tool for supervised learning problems [116,313,369,423]. Within the realm of AutoML, the integration of AutoML frameworks with RL has given rise to Automated Reinforcement Learning or AutoRL. AutoRL methods aim to automate the design of RL algorithms [29,67,212, 280], their hyperparameters [165, 395, 404], the policy/neural network [129, 266], or elements of the environment (e.g., state representation, reward, curriculum[1]) [89,110, 112, 120, 149, 371], and other components, enabling generalization across a broader range of problems and reducing the reliance on manual design processes that do not scale well. So, in addition to a better performance or generalization, AutoRL research also looks to reduce design costs [296].

Broadly speaking, AutoRL methods employ a double-loop algorithm consisting of an inner loop, where an RL agent learns policies for different environments, and an outer loop, where an AutoRL optimizer (might be also described as meta learner [175]) changes elements of the agent such as the neural network architecture, algorithm hyperparameters, or the structure of the loss function. This is illustrated in Figure 2-2. For example, we consider the case in which the AutoRL optimizer provides, fully or partially, the loss function utilized by the agent. In this case, the agent uses the provided loss function to find a good policy for the task at hand. Then, the learning curve for this process and the experience collected is passed to the AutoRL optimizer which, in turn, leverages that information and other metrics to guide its search for better loss functions for the agent. In that sense, the interface between the meta learner and the agents plays a crucial role in AutoRL frameworks, as it encodes all learning signals between them.

An interesting research direction in AutoRL is learning new algorithms or loss functions; many of the advances in RL to particular aspects of robustness (e.g., generalizability, stability) have come through algorithmic innovation [23, 69, 152, 186]. While RL algorithms have been traditionally designed by human experts, recently, several lines of work propose to view RL algorithms as tunable objects that can be

[1] A curriculum consists of a sequence of environment configurations or scenarios with increasing difficulty that the agent sees during training in order to learn more effectively.

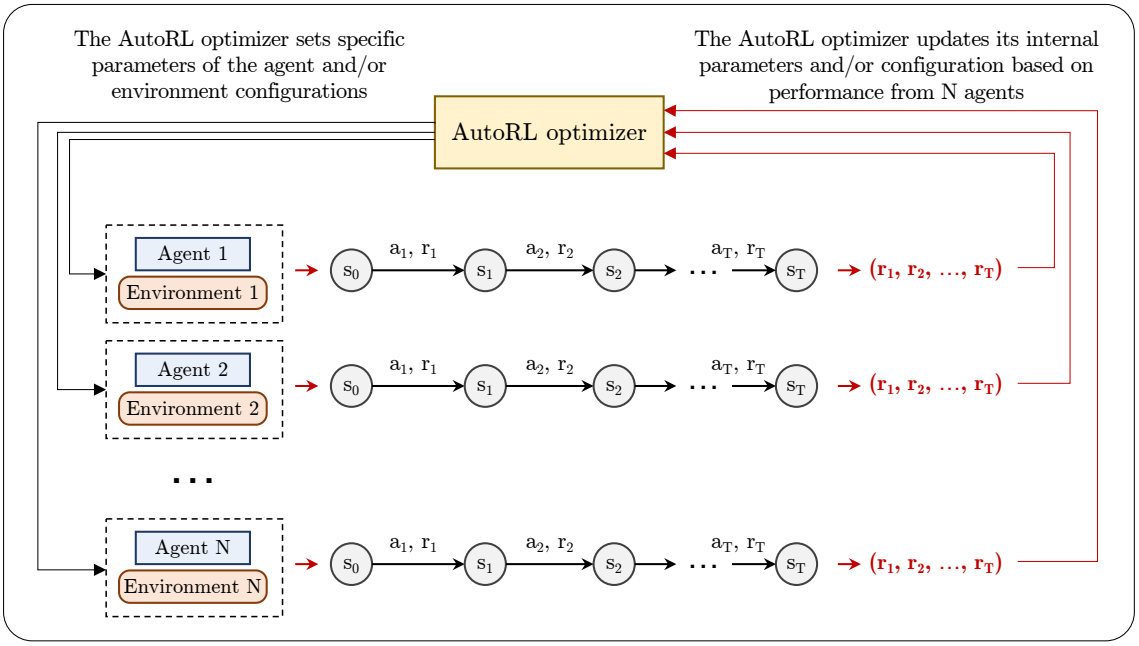



**Figure 2-2:** A generic AutoRL framework. A set of $N$ different agents, possibly interacting with different environments, attempt to learn good policies for their respective setups. Simultaneously, an AutoRL optimizer tries to learn optimal agent and/or environment configurations based on each agent's returns. The goal of the AutoRL optimizer is to improve the design of new RL agents that do better at completing given tasks, even if those have never been seen.

optimized automatically [296]. Many studies consider a fixed-form update rule which takes in meta parameters provided by a meta learner. This is the case of Houthooft et al. [176], who propose an evolutionary strategy to learn a good policy update parameter $\hat{\pi}$ that guides policy optimization. The authors meta train the meta RL framework on several MuJoCo tasks [361] and meta test on similar versions of those tasks. The same meta train and meta test procedure is followed by Bechtle et al. [29], although a meta gradient descent strategy is adopted instead of evolutionary methods. This approach is also followed by Kirsch et al. [212], who propose a framework coined as MetaGenRL to attain higher generalization. Those are proven in test environments based on unseen MuJoCo tasks. In addition to learning an appropriate policy update rule $\hat{\pi}$, the method presented by Oh et al. [280], coined as Learned Policy Gradient, also learns a prediction update rule $\hat{y}$, i.e., the semantics of the agent's prediction. In some experiments, the discovered semantics converge towards a notion of value function. The authors claim this method is able to attain higher generalizability, as proven on the performance on unseen Atari environments after meta training on toy environments. The prediction update rule is also a focus of Xu et al. [394], who

parametrize the target to update the policy and value function and learn it via meta gradient descent. Then, Lu et al. [244] draw ideas from Mirror Learning and propose to meta learn a "drift" function that benefits learning a policy.

Despite the results of the approaches introduced in the previous paragraph, all these models are limited by fixed-form update rule, i.e., all meta learning must be "encoded" within meta parameters (e.g., $\hat{\pi}$ and $\hat{y}$). One approach to avoid relying on hand-crafted update rules is to encode them as computational graphs and define a search language to modify those graphs. This idea dates back to the field of neuro-evolution [267, 343] and genetic programming [218, 315] to discover computer code, and has been applied in the context of supervised learning [314]. Co-Reyes et al. [68] study evolving loss functions for value learning algorithms. Authors introduce a search language with 26 operators used by the evolutionary framework to form different update rules in the form of directed acyclic graphs. The optimizer explores the space of graphs by means of evolutionary strategies and keeps track of a population with improved loss functions. The same idea is also present in the work by Alet et al. [15], where a meta learner produces curiosity graphs that drive the agent's exploration by shaping the environment's reward; the agent then uses the modified reward with a fixed update rule. Then, He et al. [161] propose a method to evolve auxiliary loss functions which complements predefined standard RL loss functions.

In summary, AutoRL combines AutoML frameworks with RL to automate the design and optimization of RL components, including algorithms and loss functions. By automating RL components and exploring techniques such as meta-learning and computational graph representations, AutoRL is a promising research avenue to facilitate the wider application of RL to diverse real-world problems. It offers the potential for improved performance, generalization, and reduced design costs, and it has been proposed numerous times to improve certain aspects of RL robustness determined by the real-world challenges, such as offline RL, multi-objective rewards, or generalization. An important gap for AutoRL is that, so far, it has been proposed to address one goal at a time. While the nature of real-world problems is multi-objective and many of the investigated optimizers have some variant that incorporates multi-objective optimization, it is still not clear if AutoRL methods can be designed for multiple objectives.

## 2.3 Other domain-agnostic challenges

Besides considering the challenges of real-world RL, it is important to acknowledge that there are additional research directions in the community that study other practical considerations of RL that are also important for the progress of the field overall, such as reproducibility of results, brittleness of training, interpretability, etc. These extend beyond the complexity of the lerning setups but also contribute to the bottlenecks around real-world applications. Some of them can also emerge as critical issues that warrant attention due to their impact on a broader societal level; issues like privacy [235], fairness [262], or ethics [55] are important across many ML disciplines including RL. While in this review I focus primarily on the challenges within the scope of this work, it is important to recognize that addressing concerns related to the research activity in RL and its societal impact is key for the successful deployment of RL in the real world.

The current progress in RL exhibits a certain level of brittleness and a lack of interpretability does not contribute to improve this issue. Dulac-Arnold et al. [102], in their first paper on real-world RL, identify interpretability as one of the nine key challenges to consider. Then, Henderson et al. [162] delve into the intricacies of reproducibility, experimental analysis, and reporting within the RL domain, shedding light on the importance of robust research practices. Furthermore, studies such as those by Islam et al. [189] and Zhang et al. [405] emphasize the significance of proper hyperparameter tuning in RL and other authors such as Agarwal et al. [13] and Colas et al. [72, 73] discuss proper statistical significance testing and reporting in RL. Overall, these investigations demonstrate the crucial role that these factors play in the reliability and validity of RL research outcomes.

Addressing the impact of design choices on RL systems, Reda et al. [316] examine the effect of multiple choices made on the agent side, while Andrychowicz et al. [16] conduct a similar analysis focusing on algorithmic components. Understanding how these design choices influence the performance and generality trade-off is crucial for developing robust and adaptable RL models, as explored by Hessel et al. [168] and further investigated in Zhao et al. [410]. Moreover, the suitability of smaller-sized environments for empirical work in RL is highlighted by Ceron et al. [52], suggesting that such environments offer valuable insights and experimental opportunities. Jayawardana et al. [192] explore the impact of evaluating RL models on individual Markov Decision Process (MDP) instances, as opposed to MDP families, uncovering important implications for the generalization of RL algorithms. Lastly, Engstrom et

al. [105] emphasize the significant role played by codebases in determining the outcomes of RL experiments. They emphasize the need for well-structured and accessible codebases, as they facilitate reproducibility and contribute to the overall reliability of RL research.

## 2.4 A broad look into domain-specific successes

As introduced in the beginning of this dissertation, the successful deployment of RL in real-world domains is a significant achievement, albeit one that remains relatively scarce. In this section, I aim to explore a selection of notable success stories where RL has been effectively deployed in practical settings. It is important to note, however, that the true scope of RL success may not be fully apparent due to two major constraints: 1) companies, due to proprietary interests and competitive advantage, may not disclose all successful deployments of RL, and 2) there is a similar reluctance to publicly share instances of RL failures. Therefore, while this section does not offer an exhaustive exploration of domain-specific research, it endeavors to investigate the reported deployments, considering these limitations. The examples presented herein offer valuable insights into the capabilities, challenges, and implications of deploying RL in the real world.

One of the main roadblocks in real-world RL is training in simulation before deploying. The *sim-to-real gap* is a well-known problem that many applications face. Still, the work in [283] proved a robot could learn manipulation skills and solve a Rubik's Cube only from high-quality simulation training. The agent relies on a simple algorithm, PPO [328], to learn the policy. Training in simulation also offered the advantage to easily randomize different environment properties, which helped learning more robustly. [287] and [350] are examples that follow the same framework for autonomous driving and robot mapless navigation, respectively. However, the deployment context in both applications is more complex than solving a Rubik's Cube, therefore the authors only limit themselves to very controlled test conditions in real settings. These differences suggest that the specific real-world problem addressed plays an important role in determining the deployability of a certain DRL model. This motivates our discussion in the following parts of this section.

In some cases, agents might be able to train in the real world and not need to rely on simulations. The work in [151] focuses on the ability to learn in a real setting, specifically proposing a method for a quadrupedal robot to learn locomotion skills. By maximizing both return and entropy, the robot acquires a stable gait from scratch in

about two hours. Entropy maximization might become essential in real-world training as a means to explore more and better.

In many industries, deployment involves integration within a company's operations. A good example of an industrial deployment is Google Loon's DRL-based station-keeping mechanism [31]. The company replaced its previous controller by a DRL agent that is better at keeping balloons close to base stations. A key feature of this example is that the agent's actions do not alter the environment (i.e., wind currents) and therefore the learning process has less interactions to capture. Another interesting example of industrial application is the collaboration between Microsoft's Project Bonsai and PepsiCo to develop an RL agent to adjust the extruders during the manufacturing process of certain snacks [257]. Authors emphasize that safety and minimizing risk was an important concern during development, and modeling as many real-world phenomena that occur in the factory as possible was key to achieve the result.

Another relevant industrial deployment is Meta's Horizon [139] (formerly Facebook's Horizon), which the company used to decide when to send notifications to users. Other companies like Zynga [155] have developed notification models that stem from the ideas developed in Horizon. In these cases, the possibility to gather massive amounts of data from millions of users has made RL a successful decision-maker. This mirrors the usefulness of self-play in videogames applications such as AlphaZero [335] or AlphaStar [368]. In that sense, due the absence of hardware, videogames constitute an accessible domain in which RL has proven to have an edge. The complexity of RL agents for videogames continues to increase; Wurman et al. [390] recently introuced an RL agent that matches the best e-sport drivers in the PlayStation game Gran Turismo. The underspecification of the reward function was one of the main challenges of this work.

Another key software-based problem in which RL has recently outperformed humans is designing new matrix multiplication algorithms [111]. In this work, Fawzi et al. developed AlphaTensor, an agent that found a new algorithm to multiply 4-by-4 matrices with fewer multiplications compared to the best algorithm discovered so far, more than 50 years ago. More recently, AlphaDev discovered faster sorting algorithms thanks to DRL [251]. The research lab behind AlphaTensor and AlphaDev, DeepMind, also contributed to the development and deployment of a video compression agent [249] that achieved a 6.28% reduction in size while conserving the quality. This was achieved in collaboration with YouTube, which offered data and a platform to study the implications of the deployment.

Another important accomplishment of RL in a complex real-world problem was the implementation of an RL agent to control magnetic actuator coils inside a nuclear reactor to maintain high-temperature plasma [86]. The approach provided unprecedented flexibility and was able to generalize across a diverse set of plasma configurations, thus reducing the cost of designing for them. Another example of RL deployed in a complex industrial setting was BCOOLER [245], an agent tasked with controlling commercial cooling systems. Real-world experiments showed it was able to achieve energy savings of 10%, while dealing with complext real-world challenges such as learning from offline data and a constrained action space. RL is also participating in the process of chip design [268], specifically in the layout configuration phases.

Many of these success stories come from research teams with large amounts of computational and human expertise resources; it seems that, currently, having access to those resources is a constraint for achieving RL deployments in the real world. In addition, many of the presented works rely on high-quality training environments, both software-based and hardware-based, that allow to model dynamics accurately, to train on a wide variety of environment configurations, and to capture risk and safety feature, things that overall benefits the robustness these agents need to be deployed. When smaller-scale projects attempt to apply RL, the literature shows that the majority of the studies fail to capture robustness considerations and prioritize performance when reporting the results. RL systems and agents are tailored to concrete configurations that are sufficient to prove the usefulness of RL but fail to grant the conditions for deployment.

## 2.5 Research opportunities

After conducting the literature review on different areas of real-world RL and robustness, I identify 7 research opportunities that either domain-agnostic, domain-specific of neither research thrust currently address.

**Research opportunity 1 | Domain-agnostic research thrust**

There is little research on designing for real-world robustness outside of robotics use cases.

There is a lack of research focused on designing for real-world robustness beyond robotics use cases. While robotics has been a popular domain for exploring different aspects of RL robustness, other real-world applications in which robustness is also

important have not received sufficient attention. This research opportunity highlights the need for domain-agnostic studies that, in addition to robotics, consider diverse applications. By expanding the scope of robustness research beyond robotics, we can identify generalizable principles and techniques that contribute to the deployability of RL in a wide range of real-world problems.

**Research opportunity 2 | Domain-agnostic research thrust**

There is little research on combined aspects of robustness for real-world RL.

There is a dearth of research that investigates combined challenges of real-world RL. As presented in this chapter, existing studies often focus on individual issues and tailor solutions for those. However, in practical applications, RL systems face multiple challenges simultaneously. This research opportunity emphasizes the importance of studying the interaction and trade-offs between different aspects of robustness, and motivates the study of new methods for designing RL that take this interplay into account.

**Research opportunity 3 | Domain-agnostic research thrust**

Designing for robustness is currently human-driven, which is costly when systems scale.

Currently, the design of the majority of methods that address the real-world challenges heavily rely on human-driven approaches, which are costly and inefficient when systems scale. This research opportunity highlights the need for automated and algorithmic-based design methods that enable scalability and cost-effectiveness. By leveraging different AutoML techniques such as meta learning or evolutionary search, we can develop approaches to autonomously optimize RL systems for robustness. Such integrations would significantly reduce the manual effort and expertise required, as has been proved for some of the real-world challenges, facilitating the widespread deployment of robust RL in the real world.

**Research opportunity 4 | Domain-specific research thrust**

Research prioritizes performance over robustness, this comes as a result of excessive problem tailoring.

In many published studies, the prioritization of performance over robustness is a prevalent issue in domain-specific research thrusts, often stemming from excessive

tailoring to specific configurations or scenarios of a real-world problem. Many RL agents are designed to excel under these specific scenarios that often fail to capture all the complexity of the problem and neglect robustness to different real-world challenges. This opportunity emphasizes the importance of striking a balance between performance and robustness in domain-specific RL research. By adopting a holistic approach that incorporates robustness considerations into the design process, we can mitigate the overemphasis on performance and foster the development of better RL agents for concrete problems.

**Research opportunity 5 | Domain-specific research thrust**

The designs of RL agents for specific applications do not converge.

A significant issue in the design of RL systems for real-world problems is the lack of consensus and understanding regarding the most effective components for specific applications. The literature often lacks clear guidance on which RL components work best for a given problem, leading to a lack of convergence on standard approaches. In particular applications, one can find many different designs for state representations, action spaces, and reward functions to solve the same problem; they are simply treated as elements to be reported. This opportunity highlights the need for comprehensive investigations that explore the effectiveness and suitability of different RL components in the context of specific applications. By identifying the optimal design choices and promoting convergence on effective approaches, we can enhance the robustness of RL in real-world domains.

**Research opportunity 6 | Both research thrusts**

There is little understanding of design choices and trade-offs.

There is limited understanding of the design choices and trade-offs involved in developing robust RL systems; this encompasses both domain-agnostic and domain-specific research thrusts. The complex interplay between algorithmic decisions, agent configurations, and environments poses challenges in achieving robustness. This research opportunity emphasizes the need for comprehensive studies that effectively explore the design space of RL, considering the trade-offs between performance and robustness. By gaining a deeper understanding of these design choices, researchers and practitioners can make informed decisions and develop tailored approaches that achieve an optimal balance in real-world applications.

Research opportunity 7 | Both research thrusts

Research in RL for real-world problems does not capture the full picture of designing, implementing, and operating RL systems. The design aspect is prioritized over implementation and operation, as research generally only reports on results.

The complete process of using RL for real-world problems is often overlooked in research, with a primary focus on reporting results rather than also addressing implementation and operational aspects. This research opportunity highlights the need to shift some attention towards the practical aspects of RL deployment. Understanding the challenges associated with aspects such as implementation, integration, or maintenance is crucial for deploying RL agents in the real world. By investigating the entire lifecycle of RL systems, from design to operation, researchers can provide valuable insights, guidelines, and best practices that facilitate the adoption and effective utilization of RL in practical domains.

### 2.5.1 Chapter contributions

This dissertation delves into the research area of real-world RL. In this chapter, an extensive literature review has been conducted to investigate diverse research directions, with particular emphasis on the robustness of RL systems in real-world scenarios. To the best of my knowledge, this dissertation is pioneering in its comprehensive examination of different facets of RL robustness, specifically tailored for practical applications. It makes a substantial contribution to the ongoing endeavors aimed at enhancing the effectiveness of RL in practical settings.

The specific contributions of this chapter are the following:

**Contribution 2.1** Identified and characterized two RL research thrusts, namely domain-agnostic and domain-specific, that currently contribute to the real-world applicability of RL.

**Contribution 2.2** Conducted a comprehensive review of real-world RL literature, which revealed several areas of research that are currently unaddressed by any of the existing research thrusts.

## 2.6 Thesis Statement

Given the literature review and the research opportunities identified in Section 2.5, the problem statement of this dissertation is the following:

**To** identify and investigate ways of increasing the robustness of Deep Reinforcement Learning systems operating in real-world environments **by**:

I Identifying areas of the design process that are brittle in the context of real-world DRL,

II Developing a conceptual framework that identifies key elements that influence the process of using DRL for real-world applications,

III Developing a new domain-agnostic DRL design method that simultaneously addresses multiple real-world challenges,

IV Applying the aforementioned method to a use case on the optimization of both DRL for performance and robustness,

V Designing robust DRL agents for real-world use cases in diverse domains,

VI Conducting analyses to characterize the performance vs. robustness trade-off in all explored cases,

**Using** Systems Engineering principles and Automated Machine Learning methods

## 2.7 Overview of research opportunities addressed in each Thesis chapter

To ease following this dissertation, in Table 2.2 I provide a list of the chapters to follow and point out, for each one, the specific research opportunities identified in Section 2.5 that they address, and the goals of the Thesis statement from Section 2.6 that are fulfilled.

**Table 2.2:** Relationship of Thesis chapters, research opportunities (Section 2.5), and research statement of this dissertation (Section 2.6).

| | Research opportunity | | | | | | | Thesis statement goal | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Chapter** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **I** | **II** | **III** | **IV** | **V** | **VI** |
| 3. Roadmapping DRL for Real-world Applications | | | | | | | ✓ | | ✓ | | | | |
| 4. Designing DRL Components for Real-world Applications - A Case Study on State Space Design | | | ✓ | | ✓ | | | ✓ | | | | | ✓ |
| 5. MetaPG: Optimizing Multiple RL Objectives | | ✓ | ✓ | | | | | | | ✓ | | | |
| 6. Application of MetaPG to Optimize Performance, Generalizability, and Stability | | ✓ | ✓ | | | ✓ | | | | | ✓ | | ✓ |
| 7. Designing DRL Systems for Specific Real-world Problems | | | | ✓ | | | | | | | | | |
| 8. Case Study 1: DRL for Frequency Plan Design in Satellite Constellations | ✓ | | ✓ | ✓ | | ✓ | | | | | | ✓ | ✓ |
| 9. Case Study 2: DRL for Molecular Optimization | ✓ | | | ✓ | | | | | | | | ✓ | ✓ |

# Chapter 3

# Roadmapping Deep Reinforcement Learning for Real-world Applications

This chapter presents a roadmap for designing, implementing, and operating Deep Reinforcement Learning systems in real-world environments. It is the initial attempt on comprehensively identifying all elements that influence the interaction of a DRL system and a real-world environment. Section **3.1** presents the concept of roadmaps and motivates the need for one in the case of real-world DRL. Then, Section **3.2** covers an overview of the roadmap proposed in this dissertation and its stages. The first stage corresponds to the design process and is addressed in Section **3.3**. Then, Section **3.4** presents the implementation phase and introduces the concepts of robustness, generalization, and operability in the context of the roadmap. Lastly, the operation stage and its main implications are discussed in Section **3.5**. The chapter concludes in Section **3.6** with a summary of contributions and a mapping of the remaining work in the dissertation to the roadmap.

## 3.1 Introduction

A key question when it comes to applying DRL[1] is how to make it more accessible and understandable for real-world practitioners who do not have the resources to insightfully design or implement DRL systems. This largely contributes to the gap between both domain-agnostic and domain-specific DRL research and the deployment of DRL in the real world. In Chapter 2, I identified a lack of high-level and procedural analysis when developing new DRL agents for real-world problems, and argued that

[1]In this chapter I emphasize the focus on creating a framework for *Deep* RL, therefore I use the acronym DRL.

this is not aligned with practices oriented towards robust DRL. Although applying DRL is a process, the implementation and operation of DRL systems is usually left out of the picture in the majority of the studies; the focus is put on design. However, design exercises in the literature are mostly oriented towards concrete real-world challenges (domain-agnostic thrust) or specific subproblems of a certain application (domain-specific thrust); holistic perspectives are scarce.

In the case of real-world problems, there exists a misalignment between the system-level goals and the low-level decisions that are often the focus of the literature. Given the exponential complexity of DRL systems, it is necessary to develop a set of systematic practices and protocols that align both, as it occurs in many other domains. In this chapter I propose a first attempt on developing a comprehensive conceptual framework, which I define as the *DRL roadmap*, that addresses all elements that influence the interaction of a DRL system and a real-world environment.

Roadmaps have been widely use in many scientific fields as a way to plan and/or forecast the adoption of certain technologies [82,130]. Examples include roadmapping for nanophotonics [209] and for photopharmacology [44]. While many of the examples focus on a particular field, roadmapping can be also seen from the perspective of a certain technology or product and its development over time. In that sense, the exercise of roadmapping can be simplified to setting different goals on a time horizon when developing a certain technology [82]. I use this latter perspective to refer to a set of guidelines and protocols when developing a DRL system for a specific real-world application. I assume this process can happen in different intervals with different goals separating them. This is something that has been already studied in supervised learning [372].

### 3.1.1 The complexity of Deep Reinforcement Learning systems

Figure 3-1 reproduces the first figure of this dissertation; it is the way the DRL community represents DRL systems in a comprehensive way that is easily understood by non-experts. I call this the Level 1 representation of a DRL system. However, Figure 3-2 shows that what *a priori* looks like an interaction between two components can actually be decomposed into numerous subcomponents that together influence the outcome of this interaction; it is the Level 2 representation. One can even further decompose each component into smaller interwoven elements and realize that behind the well-known two-component representation from Figure 3-1 hides a system with

exponential complexity.

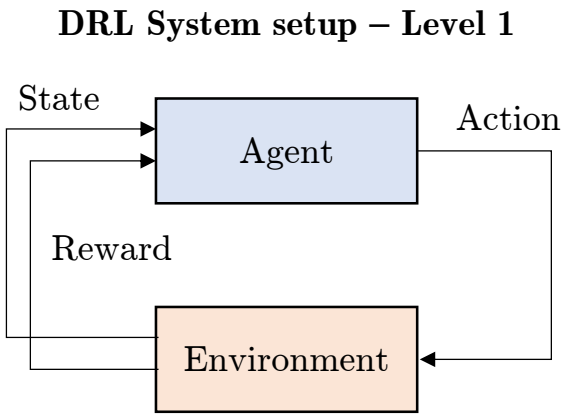



**Figure 3-1:** Level 1 representation of a Deep Reinforcement Learning system: an interaction between an agent and an environment.

Domain-agnostic DRL research has traditionally navigated this complexity by focusing on individual elements of the system. Looking at Figure 3-2, there are familiar elements that are the object of many studies in the DRL community: algorithms, reward functions, training curriculum, etc. There has been substantial research progress when it comes to each of these individual elements; one can find separate papers that address and optimize each of them (e.g., [110, 276]). However, three key differences arise when looking at a DRL system in the context of the complete system:

1. The success criteria in a real-world environment can depend on many factors that might be encoded in different parts of the system (e.g., a robot that should run fast, efficiently, and safely might need an appropriate reward function, a strong training curriculum, and an informative state representation).

2. The goal is not to optimize specific components but the interaction of all components of the system; together they determine how the agent behaves in the environment and if the success criteria are met.

3. Consequently, each component needs a conscious design decision that takes into account the whole system.

A similar set of considerations is accounted for by designers of other complex systems, for example airplanes. Airplanes are also composed of different subsystems (navigation, communications, electronics, fuel management, etc.) but the main goal is defined at a system level: fly passengers safely from point A to point B. The role of

airplane designers is then to navigate the susbsystem decomposition to identify the best design choices for each component while keeping the overall goal in mind.

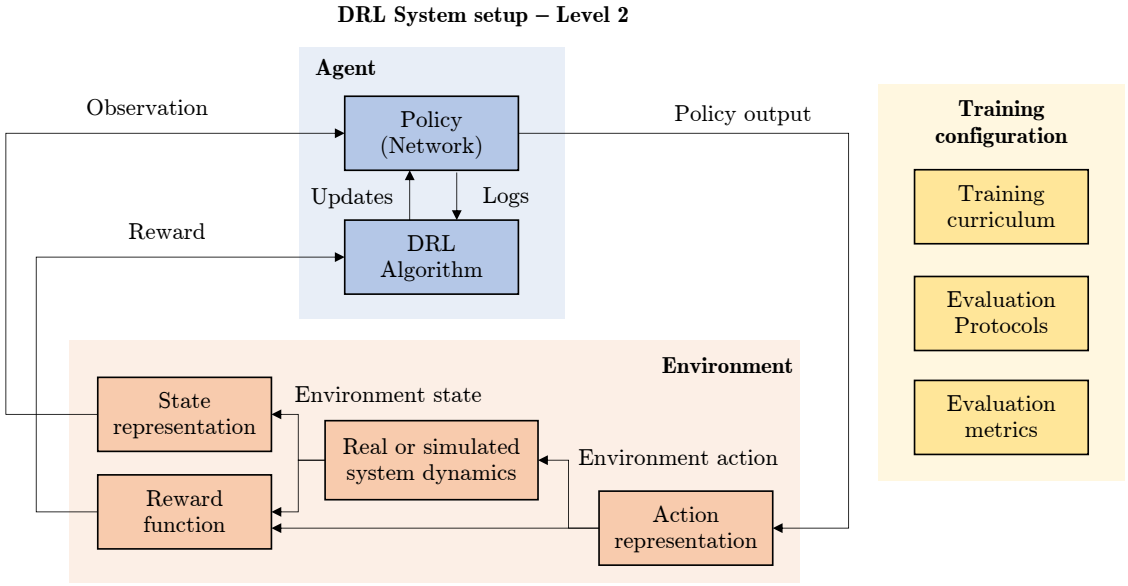



**Figure 3-2:** Level 2 representation of a Deep Reinforcement Learning system: A complex composition of interwoven elements can be observed.

### 3.1.2 The need for a Deep Reinforcement learning roadmap

When it comes to designing airplanes or other complex systems such as trains, satellites, or even certain software programs, there exist systematic design and operation processes that align the low-level efforts with the main goals of the system [76, 77]. Currently, there is no such set of practices and protocols when it comes to designing real-world DRL systems and subjectivity greatly influences development [78]. This might contribute to why there are significant bottlenecks in the road to deployment and why, currently, successfully and robustly implementing DRL is costly. It is especially relevant for domain-specific practitioners, as a DRL roadmap would help to reduce the amount of DRL expertise and computational resources needed.

In this chapter, I develop a DRL roadmap that casts a light on the different elements that are involved in the use of DRL agents in real-world contexts. I frame it as a process that connects agent design, implementation, and operation with the goal of fulfilling the success criteria defined by the real-world problem at hand. For each of the three stages, this chapter discusses key decisions, inputs, outputs, goals, related literature, and caveats to consider.

This chapter's contribution is not a recipe for "how to make DRL work for the real world"—mainly because this might be too dependent on the specific domain—but the first attempt to comprehensively identify all ingredients that are involved in that process.

I believe this roadmap is of interest to both domain-agnostic and domain-specific DRL communities. On the one hand, it introduces a *systems thinking* approach [76] for DRL, emphasizing the role of the interactions and the system-level goals when a real-world problem is in the context. On the other hand, it introduces a process to systematically design, implement, and operate DRL agents. This provides domain-specific practitioners who lack DRL expertise a deeper understanding of the elements that need to be considered when applying DRL to real-world problems and an overview of possible elements to iterate on when DRL agents cannot be deployed successfully.

## 3.2 Roadmap overview

Figure 3-3 depicts the DRL roadmap, presenting the process of developing DRL agents for real-world environments. We argue this process, regardless of the specific real-world domain considered, can be divided into three stages:

1. **Design stage** (Section 3.3) Making all decisions on the components of a DRL system. This entails deciding not only the form of each component in Figure 3-2—alongside its subcomponents—but also how the process of deciding on that form should be carried out.

2. **Implementation stage** (Section 3.4) Training the DRL agent, evaluating its fitness to the real-world environment considered, and iterating on certain design decisions of the system.

3. **Operation stage** (Section 3.5) Deploying the DRL agent in the real-world environment and managing its continuous operation over time according to the system.

### 3.2.1 The real-world task

The start point of the process is a real-world environment in which a specific task must be carried out by an autonomous agent. This can be a quadrupedal robot

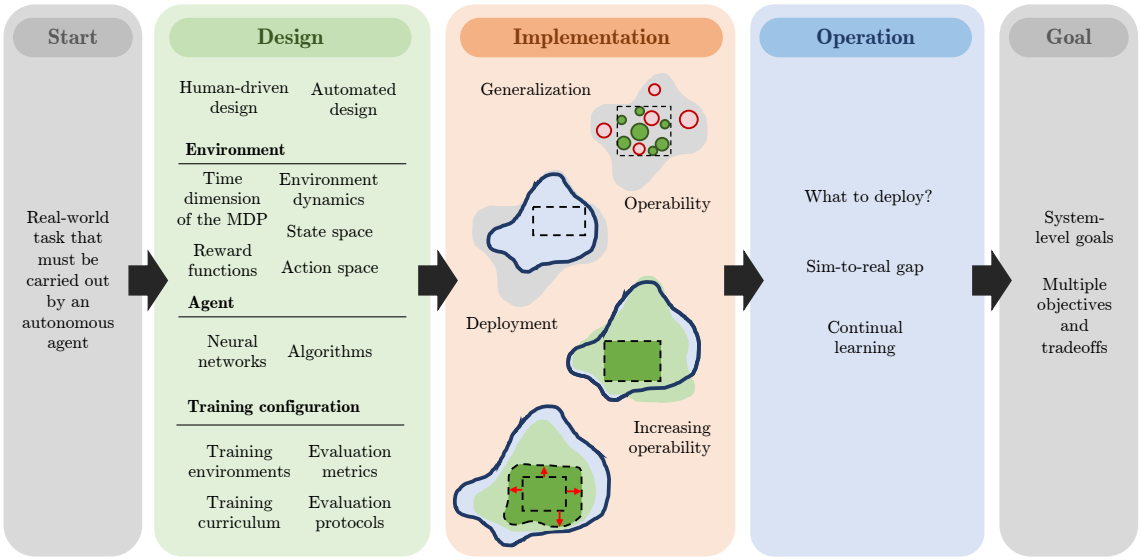



**Figure 3-3:** Overview of the roadmap for Deep Reinforcement Learning for real-world problems. Developing DRL agents for real-world applications is framed as a process that connects a start point with a goal, both defined in the context of the real-world problem. This process then has three stages: design, implementation, and operation.

walking around a construction area, a system to design a new compound targeting a specific disease, a software to decide how to allocate power in different parts of an energy grid, an autonomous vehicle transporting goods, a bot for investing in the stock market, etc. Throughout this chapter, the quadrupedal robot moving around a construction area is used as a running example (see Figure 3-4).

### 3.2.2 System-level objectives

The specification of the task entails the definition of one or more system-level criteria that determine whether the task is fulfilled and whether the interaction between the agent and the environment is successful. In our example, the operator of the robot might want it 1) to move as fast as possible in the different types of terrain found in a construction site, 2) to avoid stepping on debris or falling into holes, 3) to generalize to situations not encountered during training, and to 4) do all of that it in a cost-efficient manner. In that sense, a key characteristic of real-world tasks is that their goals are often multi-objective.

Multi-objectiveness adds a layer of complexity when evaluating a DRL agent operating in the real world, as the operator of the agent must interact with possible

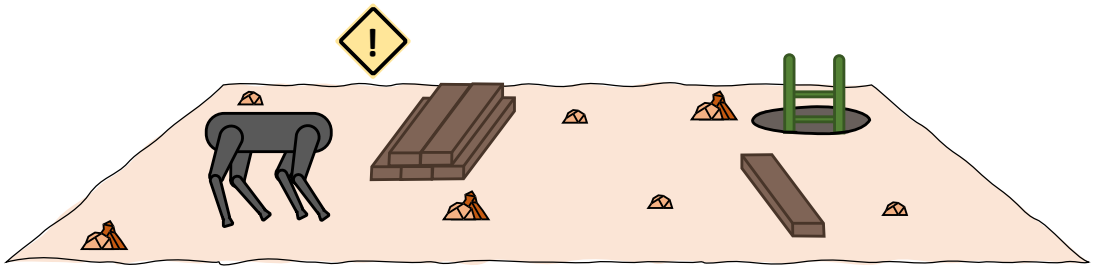


**Figure 3-4:** Support example of a robot moving around a construction area. Besides walking at an appropriate velocity through different terrains, the robot must avoid obstacles—some of them not seen during training—and be energy-efficient.

trade-offs among the different objectives. For example, our robot might be able to trade a larger speed for higher risk in its movements, or for worse generalizability. Still, one can arguably assume that, given a trained agent deployed in the environment, the operator is able to observe how well the agent meets each goal and to decide if the deployment is successful, almost in a binary fashion.

The task and its goals constitute the start and end of the process the roadmap describes, respectively. These elements are what domain-specific practitioners are usually familiar with and what motivates many domain-specific works in the literature; DRL agents are regarded as a way to connect both ends. In the following sections I provide a systematic approach to do that while touching on the important points in the middle.

## 3.3 Design stage

After scoping the task, several elements must be specified before one can train the agent. This process constitutes the design stage and entails all possible decisions the DRL system designer has influence on at both high- and low-level [16]. These decisions range from individual hyperparameters, such as the learning rate or the discount factor, to complete architectures, such as picking an suitable neural network. Some of these decisions might be straightforward, such as choosing an appropriate action space, or might require some creativity, such as finding a good training curriculum. At a high level, they can be divided into decisions connected to the environment, the agent, or the training configuration, and they can be human-driven or automated.

### 3.3.1 Manual design and automated design

The performance of a DRL system is highly-sensitive to its design. This dissertation distinguishes between two ways of making decisions on the different components of a DRL system: manual or human-driven design and automated design. They are not necessarily mutually-exclusive, as systems can be designed in combinations of both.

**Human-driven design** Generally, current DRL design is human-driven, i.e., designers directly make the choices for each of the components of the DRL system. Many of these choices come from empirical trial-and-error processes or are based on popular implementations. Those who work on domain-agnostic DRL research might possess deeper insights on what constitutes a good design choice, while domain-specific practitioners generally lack the expertise to do so. However, the latter understand well the real-world problem and therefore can further finetune specific choices to better adapt to the problem (e.g., the reward function). In this section I follow the structure in Figure 3-2 and focus on the choices that are domain-agnostic, leaving domain specificity outside of its scope.

**Automated design** Automated Machine Learning or AutoML research has seen substantial progress in the recent years, especially in the fields of hyperparameter tuning and Neural Architecture Search. AutoML methods automate the search for specific design choices by relying on different search methods such as gradient descent, population-based optimization, grid search, or even other neural network algorithms. Some recent works have applied these methods to DRL design [296]. In those cases, looking for the right design choices becomes an *outer loop* problem whereas evaluating one specific choice is seen as the *inner loop* problem. Several works have shown that using these methods to design policy networks, DRL algorithms, or generate training curricula leads to better results compared to human-driven search. However, the cost of running these algorithms increases with the complexity of the search space; there is an ongoing debate in the community on the efficiency and benefits of these methods. Still, it is a promising area of research and the reader should consider that the design of many of the elements that are described in the following paragraphs can be automated.

### 3.3.2 Environment decisions

There are several elements of the environment that require a design decision in every real-world DRL system: setting up the time discretization of the MDP, the environment dynamics to rely on, the state space, the action space, and the reward functions.

**Time dimension of the MDP** First, the real-world task must be encoded in the form of a Markov Decision Process (MDP). An often overlooked part of this step is that a time dimension must be made intrinsic to the MDP, as actions must be taken sequentially in different discrete timesteps. Some environments are sequential and discrete in nature and therefore offer a straightforward way to encode the time dimension (e.g., recommending the next movie to watch), in others time is continuous and therefore the designer must choose how to discretize it (e.g., a robot walking, our example), and lastly some environments do not possess a time dimension at all (e.g., generating a new molecule). In this latter case, the designer must decide what constitutes a timestep in the MDP. Generally, the time scale will constitute a trade-off, as picking smaller time scales might help the agent to take more precise actions but might make learning those harder.

**Environment dynamics** Another important decision is which environment dynamics to use; it usually takes the form of an operational constraint, as the real dynamics of the environment might not be available for different reasons. In those cases, designers and operators rely on simulated dynamics to train agents. While this might introduce some advantages such as gathering data faster, it entails *sim-to-real* considerations [411]; I expand more on this point in the operation stage.

The dynamics of the environment might not offer a straightforward interface to interact and learn from. As a consequence, appropriate state and action spaces must be designed; they respectively become the output and input interfaces the agent uses to interact with the environment.

**State space** The designer must decide how much information should be encoded in the state and consider the trade-offs this entails. The work in [337] provides an analysis on what makes a good representation for DRL tasks. In this chapter's example, one could choose to rely only on sensor information at the robot's joints or learn from pixels. The latter option might be less sample-efficient [355, 403] but could provide additional safety benefits. Incorporating domain knowledge is also important in the

process; many domain-specific works use hand-crafted state representations that exploit specific elements in the domain (e.g., there can be symmetry in certain robotics applications). It is worth noting that, while the MDP has a time dimension, it does not necessarily need to match with the real time dimension of the environment and therefore the state could be composed of elements of the environment that occurred in the recent past (e.g., a state formed by stacking the four most recent frames of a video game [270]). DRL research follows the progress of Deep Learning research. In that sense, for example, the state could also be encoded in a multimodal fashion [358, 362] in order to exploit information from different sources (e.g., both pixels and a feature vector). Other authors have shown that good state representations can be *learned* with contrastive learning methods [340] or by using autoencoders that capture task-relevant information [403].

**Action space** Sometimes actions might be defined intrinsically by the environment (e.g., setting a number, choosing a discrete option). Still, one can decide whether a continuous action should be discretized to ease learning or if specific sequence of actions in the real environment should be encoded together (e.g, an action could be to take three steps forward and turn right). One can even decompose the action structure in the environment by means of Hierarchical DRL [41]. This helps when the natural action space is high dimensional, combinatorial, or when large sequences of actions must be taken to meet a goal [319]. In our example, the action space could range from setting torque values in each joint to specifying higher-level actions (e.g., move, stop, turn right, etc.). Other mechanisms to deal with large action spaces have also been proposed [83, 322, 402].

**Reward functions** Rewards constitute the learning signals of the DRL system, they are a critical part of the training process. Generally, reward functions that reflect how well the agent is performing the task or how close it is from reaching the goal are preferred. In some cases, this might be naturally provided by the environment (e.g., number of points in a game, a binary signal, a delta of a quantity in two consecutive timesteps), but in other cases it might be difficult to find an appropriate reward. This has constituted the motivation behind key DRL concepts such as Inverse RL [127], Self-supervised learning for RL [156], unsupervised learning for RL [377], or entropy maximization techniques [109]. Another factor to take into account is the sparsity of the reward signal; too sparse rewards might lead to *credit assignment* problems, which some authors have already studied [17, 60, 182, 379].

One important thing to note is that one of the most relevant misalignments between DRL systems and real-world tasks is that, while the latter are usually multi-objective in terms of the goal, the former rely only on a scalar signal. Capturing a complex set of objectives in a single number is typically hard; domain-specific studies are sometimes critiqued for excessive tailoring in the reward structure. Having reward functions that are too dependent on certain parameters or coefficients might result in agents that are unable to generalize outside of the training regime. It is important not to confuse this with having multiple reward signals in the same environment (e.g., the speed and the fuel consumption of an autonomous vehicle); some system goals might be impossible to capture with a reward signal (e.g., sample efficiency, generalizability). Works that have specifically addressed having multiple reward signals in the same environment include [2, 59, 392, 396].

### 3.3.3 Agent decisions

The next set of design decisions corresponds to components of the agent that are independent from the environment: the neural networks used for the policy and other elements relying on function approximators (e.g., value functions) and the DRL algorithm.

**Neural networks** The key feature of DRL that makes it "deep" is the use of a neural network as the policy of the agent. The network should be designed to have enough representation power to map states to actions. In that sense, it should account for high-dimensional or multimodal states if necessary, and output scalar values that could be directly treated as outputs or as parameters of a specific probability distribution. In addition, many DRL systems use secondary networks that act as value networks, target networks, etc.; the design of those networks also falls into this category. Deep learning research plays a major role in finding innovative designs for policy networks; in addition to fully-connected networks, one can observe structures such as CNNs, LSTMs, attention mechanisms, or autoencoders in different DRL works in the literature. The design space of neural networks is vast, designers typically resort to popular architectures from the literature or make use of AutoML methods to automate the search.

**DRL Algorithms** The DRL algorithm is the element of the system that updates the weights of the neural networks after collecting tuples of experience, following an

iterative process. This process is usually non-stationary, as the experience tuples depend on the current policy, which is constantly changing until convergence. The complexity of DRL algorithms makes them one of the system components with more design choices and flexibility, as has been shown by several works that have attempted to automate its design [29, 67, 161, 212, 244, 280, 394]. While AutoML for DRL algorithms is one of the most promising research directions in the field [296], most of the algorithm innovations have come from human-driven design. Many human-designed algorithms have become widely popular in the DRL community and have been picked up by domain-specific works: Q-learning, PPO, SAC, TD3, DDPG, etc. Many of these algorithms are improved versions of previous algorithms in which specific shortcomings are addressed.

A full dissertation could be devoted to discuss DRL algorithms, but at a high-level, algorithms can be classified as *model-free*, if the algorithm simply learns a policy to take actions in the environment; or *model-based*, if the goal is to learn a *model* of the environment first, and then use this model to plan actions. They can also be classified as *on-policy*, if an update of the policy only takes into account the most recent experience tuples; as *off-policy*, if data collected at any point in the interaction is used; or both. They can also be classified as *policy learning* algorithms, if the goal is to learn the policy; as *value learning* algorithms, if they focus on learning a value function [348] of the environment first; or both. In addition to the specific structure of the algorithm, there are many hyperparameters than can be tuned in each of them. Some of those are algorithm-specific (e.g., hyperparameters of experience replay), and others are used by all algorithms, such as the learning rate or the discount factor $\gamma$. Deep learning algorithms also influence innovation in DRL, knowledge about specific elements such as regularizers [240] is transferred between fields.

### 3.3.4 Training configuration decisions

The last set of decisions consists of choosing how training should be carried out. This can be a dense search space, although all decisions can be primarily classified into selecting training environments and their configurations, a training curriculum, the evaluation metrics, and the evaluation protocols to follow.

**Training environments** A training environment, real or simulated, is defined by transition dynamics [348] that map the agent's actions to new environment states,

and by a specific reward function. The transition dynamics can be deterministic or stochastic, and known or unknown. If the environment is episodic, a terminal state will be defined. The agent will train by running multiple episodes in the environment; the specific number of episodes and timesteps per episode is usually set as a hyperparameter. In the cases of non-episodic environments, designers specify the total number of timesteps training will last for.

**Training curriculum** This constitutes the set of real or simulated environments the agent interacts with during training; usually there exists a hierarchy of difficulty among the different environments. The goal of using different environments in the curriculum—or defining easier tasks as part of the curriculum—is to help the agent learn faster, especially in the case of complex tasks [276]. Some designers design it empirically, others rely on methods to *learn* a curriculum [374, 375]. In the case of model-based DRL, the curriculum might consist of different datasets of experience tuples that provide different learning signals to the agent. A good curriculum can lead to aligning the agent with system-level goals such as safety. In our example, we could begin training the robot in an empty area, then add some rocks and sand to the environment, then some easy obstacles, etc. Practitioners dealing with certain real-world challenges such as non-stationarity or partial-observability might benefit from designing easier versions of such tasks to improve the robustness of the agent.

**Evaluation metrics** The goal of the agent during training is to maximize the reward. However, system-level goals might be multi-objective or hard to encode in the reward function, and therefore only looking at the accumulated reward might not be enough to evaluate a trained DRL agent addressing a real-world task. Looking at the accumulated reward or return in the training environment might indicate how good the nominal performance of the agent is, but might not be enough to evaluate its robustness. For example, to evaluate one aspect of robustness such as generalizability, the designer might come up with additional sets of environment configurations unseen during training, operate the trained policy in those environments, and use the return as a measure of how well the policy generalizes to new scenarios. In our example, we could evaluate safety by testing the trained robot on a set of environments with more obstacles and a lower risk tolerance. Other interesting real-world metrics such as sample efficiency come from directly analyzing training curves in the training environment. When it comes to the evaluation metrics, the role of the designer is to determine what to observe and the necessary tools to make that observation.

**Evaluation protocols** Evaluating a trained agent is not straightforward, the reliability of protocols followed by the majority of works in the literature has been questioned [13, 162]. First, many elements of a DRL system are stochastic in nature, which authors have addressed by running the training loop multiple times and getting confidence intervals on the different metrics. However, a recent study [13] points out that there exist better statistical measures to determine uncertainty in DRL: interval estimates, interquartile means, and performance profiles. It is also important to make sure the number of runs is adequate. This adds to previous claims on the detrimental effects of not having standardized codebases, evaluation protocols, or good ways of picking random seeds [162]. In that sense, reproducibility is another pressing problem in the community.

## 3.4 Implementation stage

After specifying the design of all elements of the DRL system, the next step of the process is to train the agent using the chosen training configuration. In case the design of one or more elements is automated, the implementation stage combines training with meta-training being run as inner and outer loop learning processes, respectively. The outcome of this stage is a trained agent that can be deployed in the real-world environment. To that end, I argue the agent must meet certain criteria that can be summarized as its abilities to 1) be robust and generalize and 2) be operated. Until meeting these criteria, the agent might be trained more than once, as certain design decisions might be revisited during this stage (see Figure 3-5).

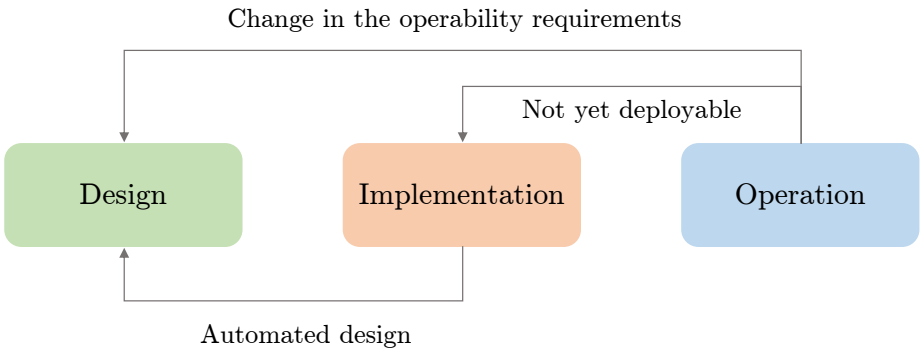



**Figure 3-5:** The DRL roadmap might not imply a sequence of steps exclusively in the forward direction, there might be circumstances that require revisiting previous phases.

### 3.4.1 Robustness, generalization, and operability

The implementation phase, in which the agent is trained, has a clear input: a fully-specified design. Then, the output of this phase is a trained agent that is ready to be deployed. Deployment is the place where, from a practical point of view, implementation should lead to. To better understand the meaning of "ready to be deployed", this section introduces three key concepts that apply to any real-world problem and concisely capture the path to be traversed until deployment and the operation phase.

**Robustness** Chapter 1 has presented robustness as one key central piece of this dissertation. It has been defined as the ability of an RL agent to satisfactorily complete a real-world problem or task across all the configurations and sets of assumptions in which the task presents itself in the real world. This entails that, in order to be deployed, a trained RL agent should be able to make decisions in the presence of any real-world challenge. The extent to which it succeeds to do so determines the extent to which this agent and the overall RL system are robust.

**System-level generalization** Also in Chapter 1, I discussed that there is a view of robustness that overlaps with the idea of generalization seen from the RL system perspective, different from the real-world challenge of generalizability. A given real-world problem can manifest in different configurations or sets of assumptions (also referred to as different subproblems). The ability to train an agent under any of these assumptions means that the RL system generalizes, and therefore can be considered robust to the problem. Given a real-world problem that can be decomposed into $N$ different subproblems (see reference to subproblems in Section 1.2.2), an RL system that generalizes can be designed in $K$ ways such that, for each subproblem, there exists at least one design variation that leads to training an agent that succeeds for that subproblem. The number of design variations $K$ can range from 1 to $N$. In our robot example, the $N$ subproblems could correspond to all possible obstacle types and layouts. Encoding all these possibilities is something almost impossible to do in practice, so usually $K$ equals 1 and the hope is that this is enough to generalize to all possible $N$ subproblems. However, operators really value generalization to the $N$ subproblems, this is something that humans do naturally.

When the system does not generalize, this can be due to three possibilities: 1) lack of interpolation generalization, when the agent fails at subproblems that can be con-

sidered "in between" two subproblems seen during training (e.g., in our example, this could mean being able to avoid small and large rocks on the ground, but fail to avoid medium-sized rocks); 2) lack of extrapolation generalization, when the agent fails at scenarios outside of the training scope (e.g., we have trained our robot in terrains with small and medium-sized rocks and observe that later it fails in scenarios with large rocks); or 3) lack of both types. Generally, it is much harder to address extrapolation problems. Figure 3-6 tries to visually summarize the effect of generalization.

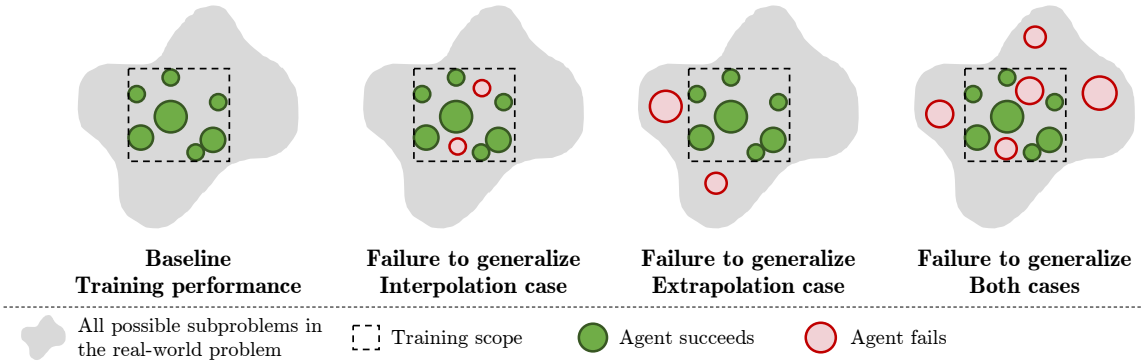



**Figure 3-6:** Overview of the different types of system-level generalization problems. After computing a baseline of the agent's performance on the training subproblems (defined inside the training scope), the agent might fail to generalize due to interpolation problems, extrapolation problems, or both.

**Operability** While a real-world problem can be defined in a generic way, as a combination of $N$ subproblems, an operator will generally be interested in a subset of $N'$ subproblems, with $N' \leq N$ (e.g., in our example, the operator might know for sure there are not large rocks on site so the agent does not need to learn to avoid them in order for it to be deployed). However, in each of the subproblems that belong to that subset, the agent must perform well according to all system-level goals. While the specific performance in each dimension might vary depending on the subproblem, a fair assumption is that the operator is able to decide, in a binary fashion, whether the trained agent succeeds or fails in a specific subproblem or scenario. The more subproblems an RL agent or an RL system is able to address, the more it generalizes, and the more robust it is. Operability refers to the ability of the agent or system to perform well in subproblems or scenarios inside the operator's area of interest (see Figure 3-7). Although operability is more important for deployment, there is a relationship between operability and robustness and generalization, as increasing generalization and robustness might also increase operability. One final note is that

there might be some scenarios that a priori are not considered by the operator, but at some point during operations, the operator becomes aware of them and adds them to the set. In that sense, the scope is dynamic.

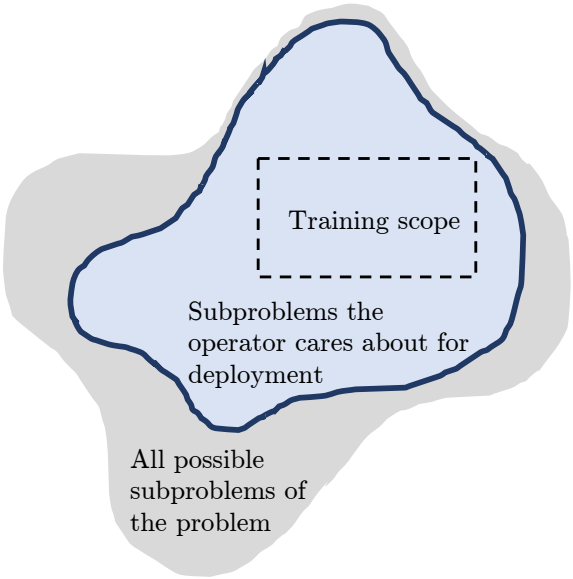



**Figure 3-7:** Representation on what might constitute the scope of the real-world problem, the scope of the region of interest in terms of operability, and the training scope.

### 3.4.2 The deployment gap

This section tries to explain why in many cases deployment of RL systems in the real world is not possible. One can use the same RL system to train multiple agents and/or run various evaluations on a trained RL agent to better understand its robustness, generalization, and operability —those tests must be designed too. Figure 3-8 represents the evaluation sucess of a trained agent in different scenarios on top of the operation area defined in Figure 3-7. The figure helps us understand the 4 possible situations we can find ourselves in after training a real-world agent for a problem (it assumes the system is only designed once, i.e., $K = 1$). To simplify these cases, the figure only discusses generalization and operability, and assumes there is no lack of generalization due to interpolation; it only focuses on generalization due to extrapolation. The figure uses the green color to refer to environment scenarios or subproblems the agent succeeds in according to the operator's binary approval that has been discussed in the previous section. These 4 possible situations are:

1. **Low generalization and low operability** In this first case there are problems that are complex and their operability entails a large number of subproblems $N$. Agents can only be trained in a small subset of the space and they do not generalize. Many proofs of concept coming from domain-specific research fall into this category.

2. **Low generalization and high operability** Many DRL systems from domain-specific studies generalize poorly but, since the real-world problem addressed is simple or well-structured, the resulting operability is high. Solving the Rubik's Cube in [283] is a good example of this kind of problems.

3. **High generalization and low operability** Even when the underlying real-world problem is complex, some agents might achieve high generalization. However, the complexity still entails low operability. Robotics applications are sometimes good examples of this case [151, 287, 350].

4. **High generalization and high operability** Finally, some agents might be able to overcome the complexity of the environment and produce policies that achieve high generalizability and high operability. There are not many examples of complex problems belonging to this category; although characterizing them using operation terminology is not usual, the agents trained in AlphaZero [335] and AlphaStar [368] would be examples in this fourth category.

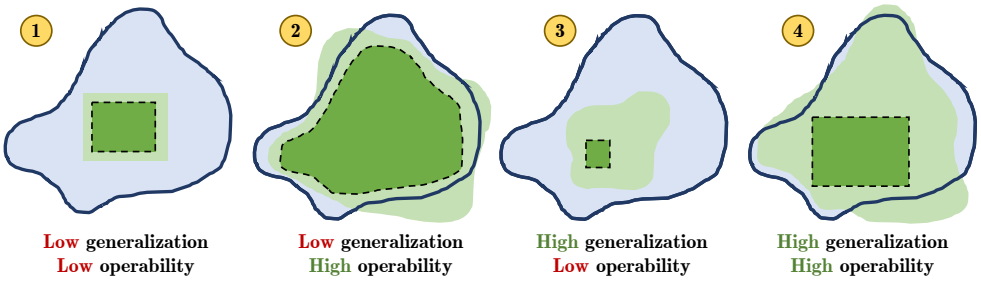



**Figure 3-8:** Comparison of 4 possible situations after training a DRL for a real-world application in terms of its generalization and operability with respect to the set of scenarios defined by an operator.

### 3.4.3 Increasing operability

The goal of the implementation stage is to produce DRL agents that can be deployed in the real-world. However, this is only possible if the operability of the agent is up

to the operator's expectations. This last section discusses four high-level ideas on how to increase operability when the outcome of the implementation stage does not match the operator's criteria. These four ideas are depicted in Figure 3-9 and can be described as follows:

1. **Extending the training scope** One can take the same DRL system and agent and extend the training space by adding new subproblems to the training data. To that end, operators must be able to incorporate them to the training curriculum, as it might not always be possible or preferred. The new training scope might also entail a different level of generalization as a consequence of changing the training distribution.

2. **Redesigning to increase generalization** An agent might generalize poorly because the environment presents certain key challenges [100] (e.g., non-stationarity, high-dimensionality, credit assignment, etc.) that are not captured by the training scope. In those cases, one can try to increase the generalization of the policy by redesigning certain components of the DRL system to be more robust against the challenges that are not being captured.

3. **Training additional agents with different designs and training scopes** In several real-world problems, the intrinsic complexity might be especially challenging for DRL agents, both their generalization and operability might be low. In those cases, one solution is to create entirely new training configurations, that might also include certain design variations, in order to address the different training scopes. This way, operability is increased by using an ensemble of agents rather than a single one. However, this approach assumes generating new training scopes is possible.

4. **Combining DRL with other decision-making methods** The complexity of the specific real-world problem might be so challenging that it is fair to assume that DRL might not be the solution to address all scenarios in the region of interest, although it is especially relevant for a subset of the subproblems. In those cases, one could rely on one or more decision-making methods to take care of the scenarios or subproblems in which the agent fails to succeed. However, finding the additional decision-makers might not be trivial.

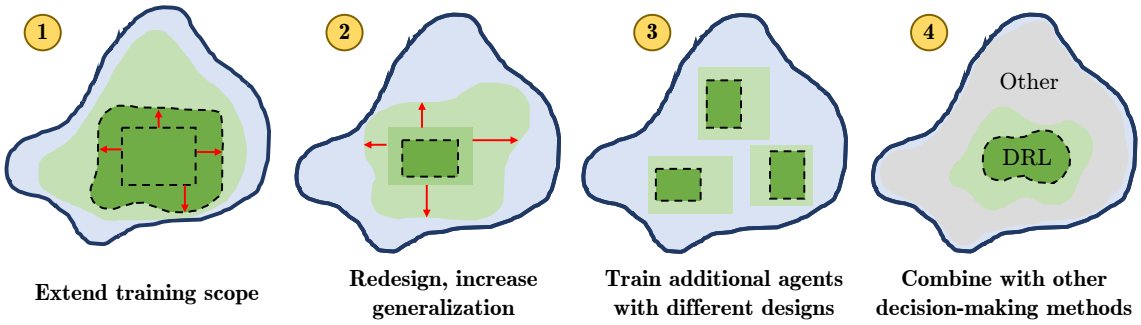



**Figure 3-9:** Four possible ways of improving the operability of a DRL agent in the context of a real-world problem.

## 3.5 Operation stage

Once the DRL system is adequately designed and the agent successfully trained, the last step of the roadmap is to deploy the agent in the real-world environment. Then, the agent operates in that environment for an indefinite amount of time. This stage is not well explored in the literature but it is equally important to consider from a practical point of view. There are certain issues that might make the agent fail, such as possible performances drops due to the sim-to-real gap if it manifests for the application at hand, or decisions on what to deploy exactly, as there might be different options after the implementation stage.

**What to deploy?** The rationale behind this question might not be obvious if only one agent is trained. However, based on the assumptions from the implementation stage, there might be cases in which we find ourselves with more than one trained agent and therefore must decide how many of these agents should be deployed, and when to use each one. The following cases support this question:

1. **When training multiple agents trading the system-level goals** If a real-world problem is multi-objective, one might be able to come up with different designs that prioritize different objectives and therefore non-dominance among objectives might occur [76]. Consequently, the operator might want to deploy all of these agents and establish conditions to switch between their policies. In our example, we could train one agent that moves faster but less safely, and another agent that prioritizes safety above anything else. In that case, we could be switching those policies depending on where in the construction site the robot is at.

2. **When training multiple agents with different designs** When following the third approach presented in Figure 3-9, after the implementation stage there will be a set of agents each of which has been trained differently. In that case, the operator has to decide which of those agents are deployed in the end, as their performance could be similar in the same scenario (i.e., overlapping performance, see Figure 3-10).

3. **When there are multiple seeds** If more than one seed is used, the exact same agent will be trained more than once, ending with one policy per seed. Answering the question of which of the runs should be deployed might not be straightforward, and the operator might want to consider doing additional tests among the agents obtained from each run. The work in [390] tangentially explores this issue.

**Sim-to-real gap** Many DRL concepts in the literature rely on simulated environments during the implementation stage due to different reasons such as the unavailability of real environments, prohibitive costs of real-world training, the potential risks involved, or the excessive time it takes to train in complex environments. However, accurately simulating the real-world environment dynamics is often challenging. Despite the best efforts, these simulations may not fully encapsulate the intricacies of the actual environments. These discrepancies invariably impact the performance and robustness of the trained agent when it operates in the real world, typically leading to a deterioration of both. Many authors, especially in the field of robotics, have attempted to study this problem in depth, as empirical tests have shown that robots that learn well during simulation do not transfer those learnings to the real system [184].

The sim-to-real gap is not just a robotics problem, but extends to other domains as well. It affects domains where training in the real world is either impossible (e.g., satellite engineering), cost-prohibitive (e.g., autonomous drones), associated with considerable risk (e.g., healthcare applications), or simply too time-consuming (e.g., molecular generation processes). If the operator finds sim-to-real problems when deploying, the process might need to go back to the design and/or implementation stages, with the aim of better aligning the DRL system and the agent with the real-world environment. In the majority of the cases this will mostly impact the training configuration and the fidelity of the training environments. The challenge is to leverage the strengths of simulations while mitigating their weaknesses, thereby facilitating

the transition of DRL agents from the simulation to the real world effectively.

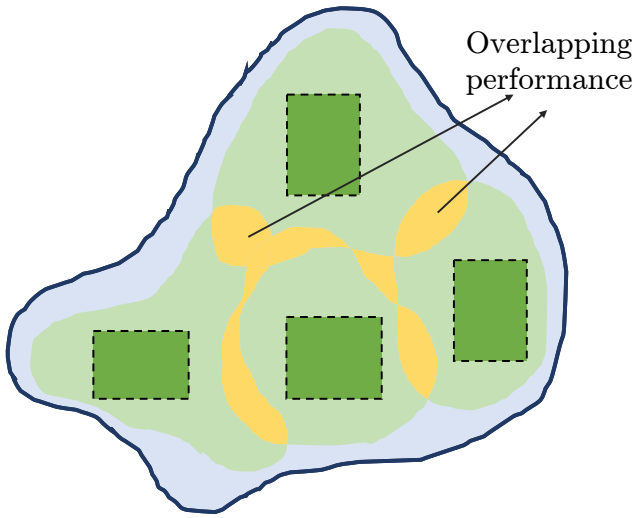



**Figure 3-10:** Situation in which different agents with different designs and training scopes are trained and, in some subproblems or scenarios, more than one agent could perform successfully, i.e., their performances "overlap".

**Continual learning** Even if one relies on real environments during training, there can be a mismatch between training and deployment performance. This can be due mainly to encountering scenarios that are not accounted for in the training configuration or due to poor evaluation protocols. Some environments might be non-stationary or partially observable, which adds a layer of complexity in the training process. While capturing all of these nuances in the design and implementation stages might be hard, DRL systems can rely on retrains or continual learning processes that ensure the agent is up-to-date with the current dynamics of the environment.

In many cases, the performance of the agent might degrade with time. In other cases, the agent might not follow episodic interactions with the environment and needs to keep learning on-the-go [375]. Continual learning in DRL is an ongoing adaptive process where the agent acquires new knowledge over time and finetunes its behavior to the changing environment. This approach is key in maintaining the agent's performance up to the operator's standards, especially in non-stationary environments where new states or actions may emerge, or the rewards associated with states/actions may change.

Scheduling these retrains, resets, and/or updates during deployment is as critical as designing an effective training process in the first place. The timing and frequency

of these updates can significantly impact the performance of the DRL system, and require careful consideration based on the dynamics and goals of the specific application. Several studies have explored the theme of continual or lifelong learning in DRL, demonstrating its importance in a variety of applications [53, 248, 275, 329].

## 3.6 Chapter summary and Contributions

This chapter has proposed a DRL roadmap as the first attempt on developing a comprehensive picture that includes all elements that influence the interaction of a DRL system and a real-world environment. This is motivated by one of the research opportunities identified in Chapter 2, which highlighted that aspects related to implementation and operation of DRL systems are usually left out of research studies. As stated in Section 3.1: this chapter's contribution is not a recipe for making DRL work for the real world, but the first attempt to comprehensively identify all ingredients that are invovled in the process.

The first section has motivated why a roadmap is necessary for real-world DRL. DRL systems can be viewed as complex systems; these are composed of subsystems and this decomposition is exponential. However, success might not depend on individual components of the system, but on their interaction. Still, the current practice is for each component to be designed in isolation. These issues are also present in many other complex systems; to navigate this complexity, there exist different systematic design and operation processes that align the low-level efforts with the main goals of the system.

Then, the complete roadmap has been presented in Figure 3-3, which is reproduced below for convenience. The roadmap connects a real-world task with the system-level goals that need to be accomplished. This is divided into three phases: design, implementation, and operation. Next, each of the phases has been characterized and discussed in detail. Section 3.3 has focused on the design phase and delved into ideas on design automation vs. human-driven design, environment-related design decisions, agent-related decisions, and decisions on training configurations.

Section 3.4 has addressed the implementation phase, which is centered around the concepts of robustness, system-level generalization, and operability. I have introduced each concept and explained how they contribute to the deployment gap that manifests in RL research. The chapter has further characterized the gap and then proposed four high-level ideas on increasing the operability of the DRL system in order to reach deployment. The chapter has concluded with a section on the operation phase, the

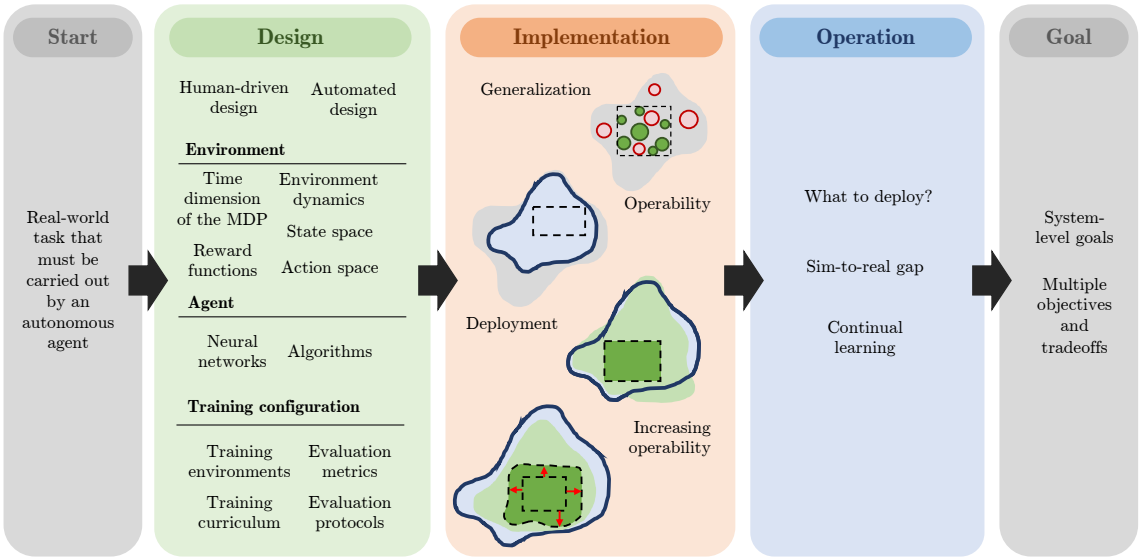



**Figure 3-3:** Overview of the roadmap for Deep Reinforcement Learning for real-world problems. Developing DRL agents for real-world applications is framed as a process that connects a start point with a goal, both defined in the context of the real-world problem. This process then has three stages: design, implementation, and operation.

last step of the roadmap. While this part is not frequently addressed in the literature, there are important aspects to consider such as choosing a set of agents to deploy, the possible presence of the sim-to-real gap, and the need of continual learning.

The specific contributions of this chapter are the following:

**Contribution 3.1** Proposed a DRL roadmap that identifies and breaks down key elements that influence the process of using DRL in real-world applications.

**Contribution 3.2** Applied the DRL roadmap to characterize the relationship between the deployment of DRL systems in the real-world and their robustness, system-level generalization, and operability.

### 3.6.1 Using the roadmap to map the rest of the dissertation

The proposed DRL roadmap also serves to summarize the focus of the remaining chapters of this dissertation. Specifically, the remaining part of the dissertation addresses multiple aspects of improving the design of DRL for real-world applications. Figure 3-11 presents the design stage of the roadmap and, for each set of chapters,

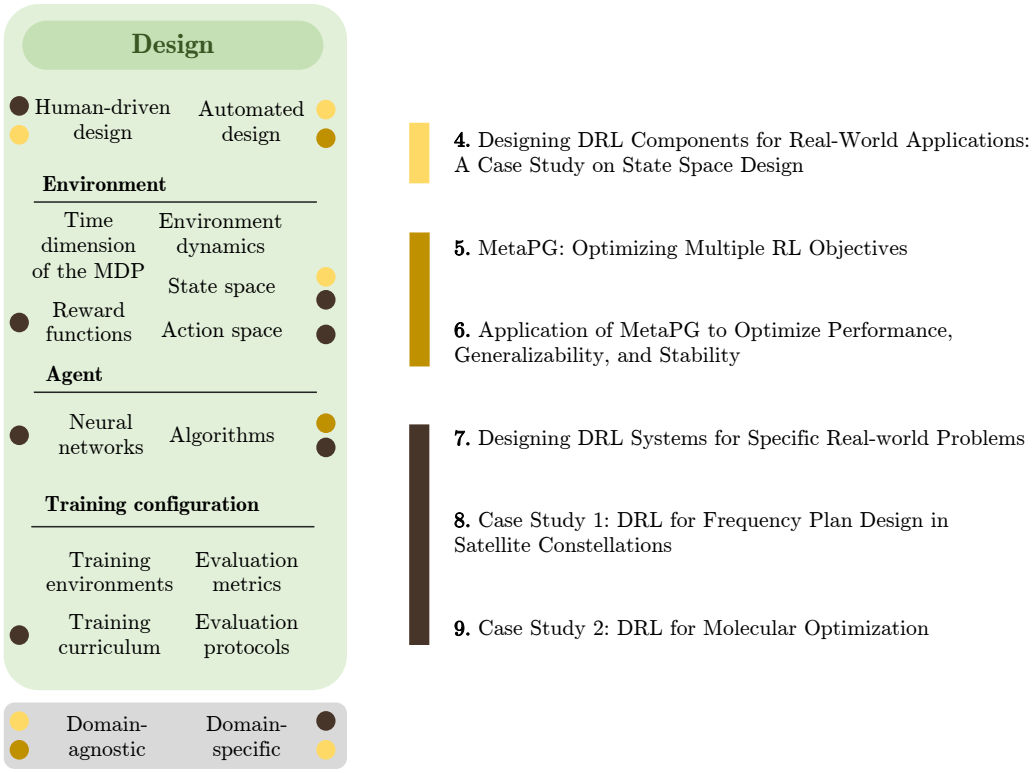



**Figure 3-11:** Mapping of the dissertation chapters on the DRL roadmap.

maps the primary focus of these chapters to provide an insight into their scope and depth. In addition, it emphasizes whether the chapters discuss domain-agnostic or domain-specific research topics, or both. Three sets of chapters with distinct focus areas are as follows:

- **Chapter 4** is a case study on improving the design of state spaces for real-world applications. It primarily focuses on one specific element of the design process and it does so from the perspective of both manual and automated design; the chapter discusses a method that combines human-driven design of features with automated selection. Although the methods and concepts discussed in this chapter are domain-agnostic, it considers a domain-specific use case, namely Traffic Signal Control.

- **Chapters 5** and **6** focus on a new domain-agnostic method for automating the design of DRL algorithms. This approach is fully automated, proposing a vast search space for actor-critic algorithms that are sufficiently representative to exclude the need for human-driven design.

- **Chapters 7, 8**, and **9** shift the focus to domain-specific examples, investigating the design of robust DRL systems for two considered use cases: frequency assignment in satellite communications and molecular optimization. Compared to the previous set of chapters, these trade the depth of the search space for breadth, exploring multiple design elements but within a significantly reduced search space. The design approach in this case is primarily human-driven.

Regardless of the focus, each chapter aims to contribute towards effectively bridging the gap between a real-world task and the system-level goals, i.e., connecting the start and the end of the roadmap.

# Chapter 4

# Designing DRL Components for Real-World Applications: A Case Study on State Space Design

This chapter provides a comprehensive exploration of feature selection in RL and its relationship with mutual information (MI) in the context of policy changes. Section **4.1** presents the problem of state space design and introduces the goals of the chapter. Section **4.2** provides an overview of the importance of feature selection in RL, its impact on policy performance, and the related literature review. Then, Section **4.3** formulates and formalizes the feature-based state space design problem. Next, Section **4.4** introduces the concept of MI and demonstrates why the policy plays a major role in the observed relevance of state features. Then, Section **4.5** takes steps toward deriving mathematical grounding on how much the MI can change when the policy changes, and what is the rate of change when the policy converges. In Section **4.6**, the analysis presented in the chapter is grounded in the Traffic Signal Control problem, a real-world use case that provides a benchmark to study feature selection. Finally, Section **4.7** summarizes the chapter and presents its main contributions.

## 4.1 Introduction

In Chapter 3, a comprehensive DRL roadmap was presented, encompassing all the important elements involved in designing, implementing, and operating DRL in real-world scenarios. Among these stages, the design phase involves a significant amount of decision-making, as various components need to be considered, and there are different

approaches to making decisions about these components. The literature provides numerous examples of DRL designs for different real-world problems. However, as discussed in Section 2.5, it can be observed that these designs often lack convergence when applied to specific applications. Researchers frequently update elements such as the state space or the reward function without a clear consensus emerging. This lack of understanding regarding the optimal DRL design for real-world problems contributes to the lack of robustness observed in many domain-specific works.

The objective of this chapter is to address this issue and propose a practical approach to enhance the design of DRL agents for real-world problems. To achieve this goal, a thorough analysis is required, and therefore, the focus is placed on a specific element of the design process depicted in Figure 3-3: the state space. The choice of the state space directly impacts the performance of the agent [316, 337] and is often determined through an empirical process [299]. When agents do not directly operate on raw sensor inputs, this process typically involves selecting features that are relevant to the problem at hand and provide the agent with the necessary information to take good actions. While selecting *a* set of features might be straightforward, choosing a *good* set of features can be challenging in many applications. This chapter delves deeper into this issue, identifies important gaps, and proposes relevant approaches to consider for addressing them. In subsequent sections, the Traffic Signal Control (TSC) problem is utilized as a use case.

Efficient feature selection has been a long-standing problem in optimization and Machine Learning [150] and has received some attention from the domain-agnostic RL community [154, 331]. Many works in the literature suggest that mutual information (MI) analysis is an effective method [247] for identifying state features that aid in predicting the reward. However, on one hand, this approach has not gained significant traction within the domain-specific community, and on the other hand, the majority of domain-agnostic studies only consider static policies (i.e., no learning) when addressing this problem. In this section, the aim is to understand the interaction between MI under policy changes and the empirical design of RL state spaces. Efficient design of state spaces is crucial, as each additional feature may require collecting additional data, which can be more intrusive for users or result in substantial hardware costs.

## 4.2 Current state space design strategies

This section provides motivation for studying state space design in RL. First, it presents related literature on state space design for RL (Section 4.2.1). Then, it discusses the issue of using an excess of features (Section 4.2.2). Finally, it explores the problem of nonconvergence using the TSC problem as an example (Section 4.2.3).

### 4.2.1 Related literature review

The question of which state representation to use for an RL agent has been approached from various perspectives. Singh et al. [337] conducted an analysis to determine the characteristics of a useful representation for RL tasks. Wang et al. [374] treated the state as an evolvable entity that follows a curriculum. Some AutoML for RL works focused on automating the design of the state and other components of the RL agent [296]. Alternatively, certain approaches involve interfacing between the environment and the policy input. Afshar et al. [10], for instance, proposed a state aggregation step to extract features based on the problem's context. Raileanu et al. [311] introduced an Upper Confidence Bound-based bandit to select image transformations for observations, while Wu et al. [387] employed canonical states to eliminate redundancy in environment observations. Contrastive learning methods [11] and tile coding techniques [348] have also been utilized for state representation. Additionally, there are works discussing feature selection in the context of specific real-world RL problems [132, 316].

Several works have addressed feature selection and dimensionality reduction in RL [237]. Some authors have investigated methods for aggregating states when approximating value functions [37, 201, 297, 298]. Other studies have employed MI to identify useful features in state spaces [154, 331]. Regularization techniques have also been used to identify favorable feature subsets [216, 236, 242], with some methods deriving from traditional Lasso regression [157]. Other authors have proposed matching pursuit methods [198, 292]. Furthermore, entropy-based dimensionality reduction during learning has been employed as an alternative approach [294, 354]. Lastly, RL itself has recently been studied as a mechanism for feature selection [238].

### 4.2.2 Excess of features

When constructing hand-crafted features for RL agents, a common approach is to include as many features as possible, with the hope of providing the agent with more

information and achieving better performance. However, it has been observed that providing an excess of features to the agent can actually degrade performance. This effect is demonstrated in three different robotics environments from PyBullet [75]. The results of an experiment conducted in the *Hopper*, *Ant*, and *Humanoid* environments are shown in Figures 4-1, 4-2, and 4-3, respectively. In this experiment, the default state space provided by the environments' codebases is taken, which includes features encoding positional and inertial information of the robot body and its joints. Groups of features are then masked out one at a time, including XYZ position of the body, XYZ velocity of the body, roll, pitch, position of the joints, velocity of the joints, and robot-ground contact features. Proximal Policy Optimization (PPO) [328] is used to train seven different agents from scratch, each using a different mask, and the resulting performance is compared with an eighth agent that uses all features (labeled as *None*). Besides the mask, each agent uses the same configuration and is trained using five different random seeds (see Appendix A.1.1).

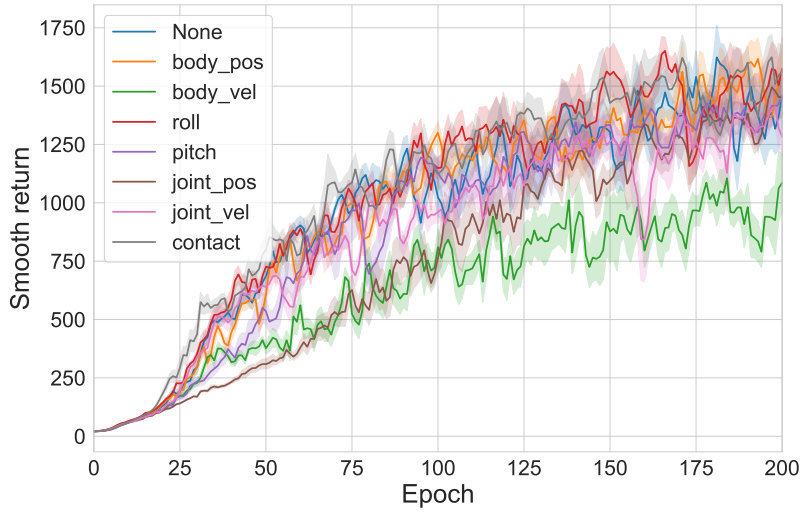



**Figure 4-1:** Learning curves when masking out different groups of features or *None* in **PyBullet Hopper**. Results from 5 different random seeds and 95% CI are shown.

The experimental results reveal that using all features does not necessarily provide any additional advantage, and in some cases, it can lead to worse performance. Removing a group of features from the state representation may have an impact on the learning curve, but the agent can still learn a good policy without relying on those features. For example, masking out the positions of the joints in the *Hopper* environment does not significantly hinder the learning process. This suggests that defaulting to using as many features as possible may lack performance benefits, which

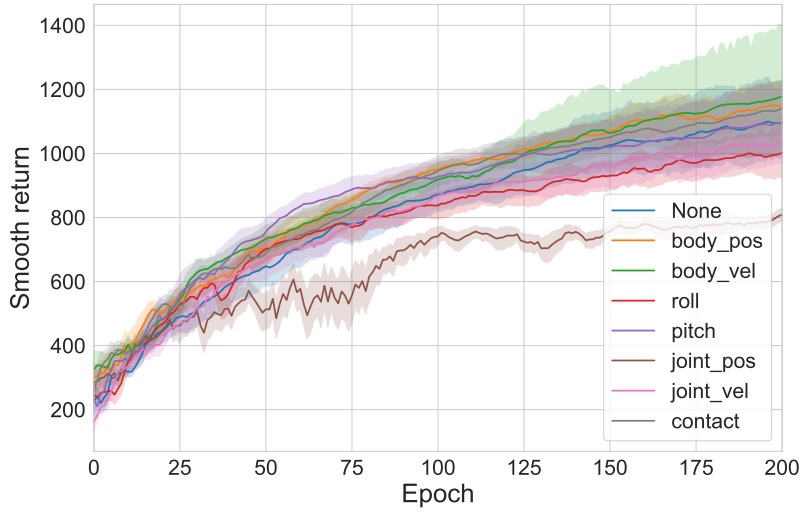



**Figure 4-2:** Learning curves when masking out different groups of features or *None* in **PyBullet Ant**. Results from 5 different random seeds and 95% CI are shown.

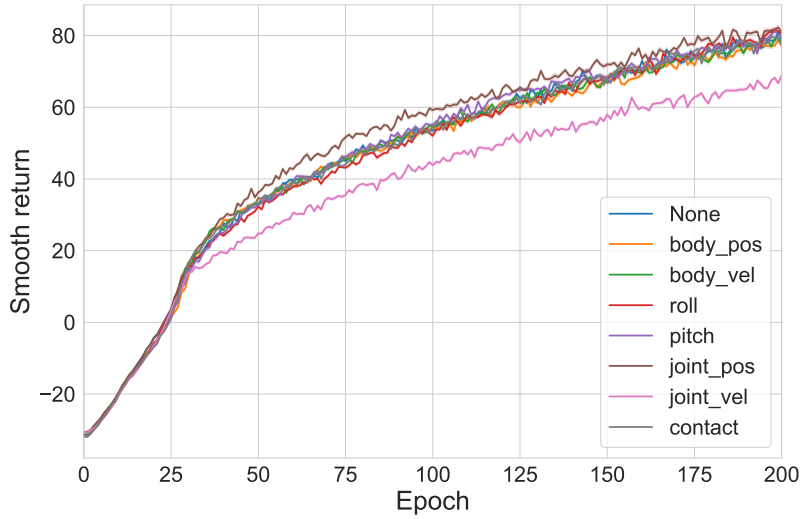



**Figure 4-3:** Learning curves when masking out different groups of features or *None* in **PyBullet Humanoid**. Results from 5 different random seeds and 95% CI are shown.

may also have practical implications, such as the need for more hardware and sensors in real-world setups.

Furthermore, the experiment highlights that the features that are beneficial or detrimental to performance vary across different environments. For instance, masking out the XYZ velocity of the body from the state representation negatively affects performance in *Hopper* but leads to the best average result in *Ant*. This indicates

that designing the state space based on similar environments, as is commonly done in domain-specific literature, may result in suboptimal feature sets.

In conclusion, defaulting to using as many features as possible or blindly copying the representation from similar environments is not always advantageous in practical RL applications. However, determining the optimal feature set is not straightforward, especially when the number of available features is large in complex environments. Empirical search can be employed when the number of features is small, but exploring the feature space becomes challenging when dealing with a large number of features. Examples of these large feature spaces include the StarCraft environment, which has up to 512 unitary features in addition to the map data [368], or RL-based power grid control, with an observation space of nearly 5,000 features [94].

### 4.2.3 Non-convergence in literature

Another important observation is that consensus on state representations does not naturally emerge in many domain-specific communities. To further highlight this problem, the Traffic Signal Control (TSC) problem [169, 382, 417] is considered. The TSC problem consists of controlling a set of traffic lights at a road intersection in order to optimize traffic flow. Even though there are more than 160 peer-reviewed studies proposing new RL approaches to solve the problem or aspects of it [278], and many high-fidelity simulators have been developed [241, 263, 407], the deployment of RL systems in real road networks is still a path to be traversed. This problem has also been a use case in other studies focusing on empirical aspects of RL design [192].

Table 4.1 presents different state spaces for the TSC problem found in the literature. Specifically, it looks at the most recent papers identified in the systematic literature review from [278]. The vast majority of the methods propose feature-based state spaces encoding features from one or more of six different categories: phase, occupancy, position, speed, waiting time, and spatial encoding.

While none of these studies defaults to using as many features as possible, it can be observed that there is little overlap between state spaces considered in different works, with more than 10 feature sets being proposed. In addition, multiple works that use the same feature might encode it differently (e.g., some authors encode the speed as average speed in the lane, while others encode the speed of each individual vehicle). As a result, when designing a new RL method to address the TSC problem, it is not clear which features it should rely on to achieve the best performance.

**Table 4.1:** Feature utilization in the state space for multiple papers using RL for Traffic Signal Control. Feature categories include the current phase, the occupancy of each lane in the intersection, the position of the vehicles in the lane, the vehicles' speed, and their waiting time. In addition, some state spaces are encoded as images of the intersection seen from above.

| | Feature categories observed for each lane in the intersection | | | | | |
|---|---|---|---|---|---|---|
| Reference | Phase | Occupancy | Position | Speed | Waiting time | Spatial encoding |
| [65, 194] | | ✓ | | | ✓ | |
| [173, 380, 381, 415] | ✓ | ✓ | | | | |
| [18, 214, 320, 321] | | ✓ | | | | |
| [317] | ✓ | ✓ | ✓ | ✓ | | ✓ |
| [330] | | ✓ | ✓ | ✓ | | ✓ |
| [57, 417] | | ✓ | | ✓ | | |
| [181, 333] | ✓ | | ✓ | ✓ | | |
| [143] | | ✓ | | | | ✓ |
| [169] | | | | | ✓ | |
| [140] | | | ✓ | ✓ | | ✓ |
| [206] | | ✓ | | | | |

## 4.3 Notation and Formalism

A sequential decision-making task is denoted as $T$ and modeled as a Markov Decision Process (MDP) with a state space $\mathcal{S}_T$[1]. Each state $s \in \mathcal{S}_T$ represents an array of $N$ features, expressed as $s = [s_1, s_2, ..., s_N]$. These features are predetermined by the MDP designer and provide relevant information about the task $T$. The performance of an RL agent trained to solve task $T$ using the MDP is captured by the random variable $X^T_{\mathcal{S}_T}$. Each training session produces a realization of $X^T_{\mathcal{S}_T}$.

As demonstrated in Section 4.2.2, the performance of the RL agent depends on the MDP it is trained on, including the choice of the state space. Hence, additional state spaces denoted as $\mathcal{S}_m$ are explored, where $m \in [0, 1]^N$ and $m$ is not the zero array $\mathbf{0}$. Each $\mathcal{S}_m$ is defined as $s_m = s \odot m = [s_1 m_1, s_2 m_2, ..., s_N m_N]$, where $s$ represents any state from the MDP modeled by $\mathcal{S}_T$. Here, $m$ acts as a mask over the $N$ features provided by the MDP designer. The performance of a specific mask is represented by the random variable $X^T_{\mathcal{S}_m}$.

The objective is to find the mask $m^*$ that maximizes the expected performance $\mathbb{E}[X^T_{\mathcal{S}_m}]$. In other words, $m^*$ is determined as $\underset{m}{\text{argmax}}\, \mathbb{E}[X^T_{\mathcal{S}_m}]$. Notably, when $m = \mathbf{1}$, the original state space $\mathcal{S}_T$ is considered, allowing for the possibility that using all

[1] In some cases, $\mathcal{S}_T$ is simplified as $\mathcal{S}$.

features yields the best outcome. Evaluating all $2^N - 1$ masks is impractical for large values of $N$. Instead, when given a budget of $M$ masks $\boldsymbol{m_1}, ..., \boldsymbol{m_M}$ for evaluation, the focus is on obtaining an estimate of the best mask $\hat{m}^*$ that minimizes the expected regret $\mathbb{E}[X^T_{\mathcal{S}_{m^*}} - X^T_{\mathcal{S}_{\hat{m}^*}}]$.

## 4.4 Feature selection via mutual information under policy changes

The previous sections have highlighted the potential for improvement in current state space design strategies for RL. While the problem of identifying the number and selection of features has been extensively studied in supervised learning (SL) problems [150], the use of mutual information (MI) analysis [247,364] has emerged as a widely-adopted solution in these cases, allowing for quantification of the relationship between specific features and the class being predicted. While feature selection via MI in RL has been explored to some extent in the literature [154,331], existing studies primarily focus on scenarios where the policy remains fixed. To the best of my knowledge, there is a lack of research on feature selection via MI that takes into account the learning context. This section introduces the concept of why a fixed-policy context might limit the understanding of which state features are useful.

### 4.4.1 Expressing mutual information as a function of the agent's policy

The MI between two discrete random variables $X$ and $Y$ is defined as

$$I(X,Y) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} P_{XY}(x,y) \log \left( \frac{P_{XY}(x,y)}{P_X(x) P_Y(y)} \right) \tag{4.1}$$

In the case of SL, the variables $X$ and $Y$ correspond to features and the class to be predicted, respectively. In the case of RL, $X$ and $Y$ represent state features and rewards, respectively. In both cases, MI can also be computed for pairs of features, which is particularly relevant for identifying redundancies.

When considering RL, there are two important differences with respect to SL. Firstly, due to the sequential nature of RL problems, state-reward samples may not be independent, whereas in SL, feature-class samples are generally independent. Secondly, in RL, the distribution of states and rewards can change as the agent learns,

whereas in SL, there is usually no update of the data resulting from learning a model. This distinction arises from the fact that states and rewards are connected by the agent's policy. Most studies in the literature compute MI under the optimal policy or using datasets of experience collected by a specific policy, disregarding the fact that policies change and, consequently, so do state-reward distributions.

To gain a better understanding of this, let's apply Equation (4.1) to compute the MI between the state $S$ and the reward $R$ in a MDP with a discrete state space $\mathcal{S}$, a discrete action space $\mathcal{A}$, and a discrete set of rewards $\mathcal{R}$:

$$I(S,R) = \sum_{r\in\mathcal{R}}\sum_{s\in\mathcal{S}} P_{SR}(s,r)\log\left(\frac{P_{SR}(s,r)}{P_S(s)P_R(r)}\right) \tag{4.2}$$

The joint distribution $P_{SR}(s,r)$ can be expressed as:

$$P_{SR}(s,r) = P_S(s)P_{R|S}(r|s) = P_S(s)\sum_{a\in\mathcal{A}}\pi(a|s)P(r|s,a) \tag{4.3}$$

As observed, the above expression depends on the policy $\pi$. Therefore, the MI between $S$ and $R$ given by policy $\pi$ can be defined as:

$$I(S,R;\pi) = \sum_{s\in\mathcal{S}}\sum_{r\in\mathcal{R}} P_S(s)\left(\sum_{a\in\mathcal{A}}\pi(a|s)P(r|s,a)\right)\log\frac{P_S(s)\sum_{a'\in\mathcal{A}}\pi(a'|s)P(r|s,a')}{P_S(s)\left[\sum_{s'\in\mathcal{S}}P_S(s')\left(\sum_{a''\in\mathcal{A}}\pi(a''|s')P(r|s',a'')\right)\right]} \tag{4.4}$$

### 4.4.2 The impact of changing policies: A toy example

To further illustrate the effect of policy changes on MI between state features and rewards, we can examine a simple toy example. Consider a MDP with a two-feature state space consisting of four possible states and a two-action space ($a_1$ and $a_2$). The MDP has deterministic rewards for each state-action pair, as shown in the left side of Figure 4-4. For simplicity, we focus on a single time-step interaction and assume a policy takes action $a_1$ with probability $\pi(a_1)$ regardless of the state (taking action $a_2$ with probability $1-\pi(a_1)$).

Then, an alternative definition for the MI between two random variables $X$ and $Y$ is the following

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{4.5}$$

where $H(\cdot)$ corresponds to the entropy of a random variable and $H(\cdot|\cdot)$ is the conditional entropy. Therefore, the MI between states and rewards can be computed

**Original MDP**

| State ($F_1$, $F_2$) | Reward $a_1$ | Reward $a_2$ |
|---|---|---|
| 0, 0 | 0 | 0 |
| 0, 1 | 0 | 1 |
| 1, 0 | 0 | 0 |
| 1, 1 | 0 | 1 |

**MDP Feature 1**

| Feat. 1 | Action | P(R=0) | P(R=1) |
|---|---|---|---|
| 0 | $a_1$ | 1 | 0 |
| 0 | $a_2$ | 0.5 | 0.5 |
| 1 | $a_1$ | 1 | 0 |
| 1 | $a_2$ | 0.5 | 0.5 |

**MDP Feature 2**

| Feat. 2 | Action | P(R=0) | P(R=1) |
|---|---|---|---|
| 0 | $a_1$ | 1 | 0 |
| 0 | $a_2$ | 1 | 0 |
| 1 | $a_1$ | 1 | 0 |
| 1 | $a_2$ | 0 | 1 |

**Figure 4-4:** Complete MDP for the toy example and partial MDPs that originate if we only consider each feature separately. The rewards become probabilistic when focusing on only one feature.

as $H(S) - H(S|R)$. Looking at the entropy of the state $S$, one can immediately observe two extreme cases. On one hand, if all states are the same, then $H(S) = 0$ and $MI(S, R) = 0$, since, by definition, $MI(S, R) \geq 0$ and $H(\cdot) \geq 0$, i.e., both MI and entropy are nonnegative quantities. On the other hand, in case there are $N_S$ equiprobable states, $H(S)$ equals:

$$H(S) = -\sum_{s\in\mathcal{S}} P_S(s) \log P_S(s) = -\frac{1}{N_S}\sum_{s\in\mathcal{S}} \log \frac{1}{N_S} = -\log \frac{1}{N_S} = \log N_S \tag{4.6}$$

This corresponds to the maximum value the MI can attain. This also holds whether the MDP corresponds to the state space with both features or just one of the two features, the difference will be the value of $N_S$. Figure 4-4 represents the complete MDP of this toy example and each of the two MDPs that originate if only one of the two features is considered. Since the goal is to compute the MI between each feature and the reward, the MDPs must be considered separately. As observed in the figure, the rewards might become stochastic instead of deterministic.

Whether the full or the partial MDP is considered, the term $H(S|R)$ to compute the MI is computed as

$$\begin{aligned} H(S|R) &= -\sum_{r\in\mathcal{R}}\sum_{s\in\mathcal{S}} P_{S|R}(s|r)P_R(r) \log P_{S|R}(s|r) = \\ &= -\sum_{r\in\mathcal{R}}\sum_{s\in\mathcal{S}} P_{R|S}(r|s)P_S(s) \log \frac{P_{R|S}(r|s)P_S(s)}{\sum_{s'\in\mathcal{S}} P_{R|S}(r|s')P_S(s')} \end{aligned} \tag{4.7}$$

In equation (4.7), the term $P_{R|S}(r|s)$ equals $\sum_{a\in\mathcal{A}} \pi(a|s)P(r|s, a)$. As mentioned previously, in this toy example, two assumptions with respect to the policy are made, 1) there are only two actions, $a_1$ and $a_2$, and 2) the probability of taking either action is independent of the state, i.e., $\pi(a_1|s) = \pi(a_1), \forall s$. Therefore, $\pi(a_2) = 1 - \pi(a_1)$. Then, assuming the state is equiprobable (and hence $H(S) = \log N_S$), Equation (4.7)

can be simplified as follows:

$$H(S|R) = -\sum_{r\in\mathcal{R}}\sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}}\pi(a|s)P(r|s,a)P_S(s)\log\frac{\sum_{a'\in\mathcal{A}}\pi(a'|s)P(r|s,a')P_S(s)}{\sum_{s'\in\mathcal{S}}\sum_{a''\in\mathcal{A}}\pi(a''|s')P(r|s',a'')P_S(s')} =$$
$$= -\frac{1}{N_S}\sum_{r\in\mathcal{R}}\sum_{s\in\mathcal{S}}g(s,r)\log\frac{g(s,r)}{\sum_{s'\in\mathcal{S}}g(s',r)} \tag{4.8}$$

where $g(s,r)$ corresponds to $\pi(a_1)P(r|s,a_1) + \pi(a_2)P(r|s,a_2)$. Then, for each of the two features, one can obtain the curves shown in Figure 4-5 by considering the corresponding partial MDP from Figure 4-4 and computing the MI as

$$I(S,R;\pi) = \log N_S + \frac{1}{N_S}\sum_{r\in\mathcal{R}}\sum_{s\in\mathcal{S}}g(s,r)\log\frac{g(s,r)}{\sum_{s'\in\mathcal{S}}g(s',r)} \tag{4.9}$$

This figure demonstrates the dependency of MI on $\pi$, as we simulate a sweep through

| State ($F_1$, $F_2$) | Reward $a_1$ | Reward $a_2$ |
|---|---|---|
| 0, 0 | 0 | 0 |
| 0, 1 | 0 | 1 |
| 1, 0 | 0 | 0 |
| 1, 1 | 0 | 1 |

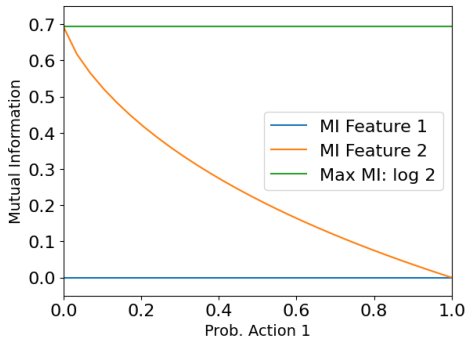



**Figure 4-5:** Mutual information change as a function of the change in the policy given by the probability of taking the first action in the toy example.

different values of $\pi(a_1)$. It shows that the MI between the second feature and the reward, computed using Equation (4.4), is maximal ($\log 2$) when action $a_1$ is never taken, and minimal when the agent always takes action $a_1$. In contrast, the first feature is irrelevant and consistently has zero MI with respect to the reward, regardless of the policy.

This toy experiment highlights that as the policy changes, the MI between state features and the reward can also change. Consequently, certain features may appear more or less relevant for predicting the reward.

## 4.5 Deriving mathematical intuition using neural contextual bandits

After observing that the MI is dependent on the policy and therefore features can be deemed more or less important without changes in the MDP, this section begins taking steps towards a better understanding of these issues by providing mathematical intuition on how much MI can change. To take the simplest form of learning paradigm, it utilizes neural contextual bandits [338] and focuses on single-timestep interactions.

### 4.5.1 How much can mutual information change between two similar policies?

Let's consider two policies $\pi_1$ and $\pi_2$ for the same MDP, which consists of a discrete set of states $\mathcal{S}$, a discrete set of actions $\mathcal{A}$, and a reward function of the form $P(r|s,a) \rightarrow [0,1]$. The goal is to derive a bound for the quantity $|I(S,R;\pi_1) - I(S,R;\pi_2)|$. To that end, the following is assumed:

**Assumption 1** The difference between both policies is considered small, i.e., $|\pi_1(a|s) - \pi_2(a|s)| \leq \epsilon_\pi$, $\forall\, a, s$, with $\epsilon_\pi \in (0,1)$.

First, Equation (4.4) can be simplified to

$$I(S,R;\pi) = \sum_{s\in\mathcal{S}} P_S(s) \sum_{r\in\mathcal{R}} \left[ \left( \sum_{a\in\mathcal{A}} \pi(a|s) P(r|s,a) \right) \log \frac{\sum_{a\in\mathcal{A}} \pi(a|s) P(r|s,a)}{\sum_{s'\in\mathcal{S}} P_S(s') \left( \sum_{a\in\mathcal{A}} \pi(a|s') P(r|s',a) \right)} \right] \tag{4.10}$$

and further simplified as follows

$$I(S,R;\pi_1) - I(S,R;\pi_2) = \sum_{s\in\mathcal{S}} P_S(s) \sum_{r\in\mathcal{R}} \alpha(s,r) \tag{4.11}$$

where

$$\alpha(s,r) = \beta(s,r,\pi_1) \log \frac{\beta(s,r,\pi_1)}{\sum_{s'\in\mathcal{S}} P_S(s')\beta(s',r,\pi_1)} - \beta(s,r,\pi_2) \log \frac{\beta(s,r,\pi_2)}{\sum_{s'\in\mathcal{S}} P_S(s')\beta(s',r,\pi_2)} \tag{4.12}$$

with $\beta(s,r,\pi) = \sum_{a\in\mathcal{A}} \pi(a|s) P(r|s,a)$. By definition, given $0 \leq \pi(a|s) \leq 1$, $0 \leq P(r|s,a) \leq 1$, and $\sum_{a\in\mathcal{A}} \pi(a|s) = 1$, the inequality $0 \leq \beta(s,r,\pi) \leq 1$ holds. Analyzing the function $x \log x$ between 0 and 1, one can see that $0 \log 0 = 1 \log 1 = 0$, and the function is negative in that interval with a minimum at $x = 1/e$. Similarly, taking into account that $\sum_{s\in\mathcal{S}} P_S(s) = 1$, the expression $0 \leq \sum_{s\in\mathcal{S}} P_S(s)\beta(s,r,\pi) \leq 1$ holds.

Equation (4.12) can be rewritten as follows

$$\begin{aligned}\alpha(s,r) =&\beta(s,r,\pi_1)\log\beta(s,r,\pi_1)-\beta(s,r,\pi_1)\log\left(\sum_{s'\in\mathcal{S}}P_S(s')\beta(s',r,\pi_1)\right)\\ &-\beta(s,r,\pi_2)\log\beta(s,r,\pi_2)+\beta(s,r,\pi_2)\log\left(\sum_{s'\in\mathcal{S}}P_S(s')\beta(s',r,\pi_2)\right)\end{aligned} \tag{4.13}$$

where the terms in blue and orange will be treated separately. Regarding the former, the goal is to derive a bound for the absolute value of the expression

$$\beta(s,r,\pi_1)\log\beta(s,r,\pi_1)-\beta(s,r,\pi_2)\log\beta(s,r,\pi_2) \tag{4.14}$$

which can be rewritten as

$$\sum_{a\in\mathcal{A}}P(r|s,a)\left[\pi_1(a|s)\log\beta(s,r,\pi_1)-\pi_2(a|s)\log\beta(s,r,\pi_2)\right] \tag{4.15}$$

where $P(r|s,a)$ depends on the environment. When evaluating the inner expression $\pi_1(a|s)\log\beta(s,r,\pi_1)-\pi_2(a|s)\beta(s,r,\pi_2)$, three different cases can be observed:

1. $P(r|s,a)=0$, $\forall a$ for the $s$ and $r$ considered. In this case the inner expression can be neglected since (4.15) evaluates to zero.

2. $\pi_1(a|s)=\pi_2(a|s)=0$ for all actions with $P(r|s,a)>0$. In this case the expression also evaluates to zero.

3. $\pi_1(a|s)P(r|s,a)>0$ for some $a$. In this case the inner expression will be different than zero.

Based on the previous three cases, one can conclude that there will be a change in mutual information whenever $\pi_1$ or $\pi_2$ fulfills that $\pi(a|s)P(r|s,a)>0$ for some $a$.

**Assumption 2** $\pi_1(a|s)>0$, $\forall a$, $\pi_2(a|s)>0$, $\forall a$, and there exists at least one action $a'$ such that $\pi_1(a'|s)P(r|s,a')>0$ and $\pi_2(a'|s)P(r|s,a')>0$.

Now, to find a bound for expression (4.14), the *Mean Value Theorem* (MVT) is used. Given a continuous and differentiable function $f$ in the interval $(x,y)$, the MVT states that

$$\frac{f(a)-f(b)}{a-b}\leq f'(x)\quad\text{for all } a<x<b \tag{4.16}$$

In this case the function to consider is $f(x)=x\log x$. Expression (4.14) can be seen as $f(x_1)-f(x_2)$, where $x_1=\beta(s,r,\pi_1)$ and $x_2=\beta(s,r,\pi_2)$. The derivative of $f$ is

$f'(x) = \log x + 1$. From Equation (4.16), it follows that,

$$|f(a) - f(b)| \leq |f'(x)||a - b| \leq \max_{x \in (a,b)} |f'(x)||a - b| \tag{4.17}$$

Given that the derivative of the considered function, $f'(x) = \log x + 1$, is monotonically increasing, $f'(x)$ will have its maximum in the interval $(a, b)$ at $a$. To find precise values for the bound, the following two assumptions are made

**Assumption 3** $\sum_{a \in \mathcal{A}} P(r|s, a) \geq p_r$, i.e., the total probability of attaining reward $r$ from state $s$ is at least $p_r$. Given Assumption 2, $p_r > 0$.

**Assumption 4** For both $\pi_1$ and $\pi_2$, $\pi(a|s) \geq 1/k_a \, \forall \, a$, with $k_a \in \mathbb{R}^+$ and $k_a \geq |\mathcal{A}|$, where $|\mathcal{A}|$ is the total number of different actions. This means all actions have a certain probability of being chosen that is greater than or equal to $1/k_a$.

From these assumptions the following holds

$$\beta(s, r, \pi) = \sum_{a \in \mathcal{A}} \pi(a|s) P(r|s, a) \geq \frac{p_r}{k_a} \tag{4.18}$$

Therefore, the value for $a$ in Equation (4.17) cannot be lower than $p_r/k_a$. Now, taking into account Assumption 1, the following inequality can be defined

$$\beta(s, r, \pi_1) - \beta(s, r, \pi_2) = \sum_{a \in \mathcal{A}} P(r|s, a)[\pi_1(a|s) - \pi_2(a|s)] \leq \frac{1}{2}|\mathcal{A}|\epsilon_\pi \tag{4.19}$$

where the multiplication by $1/2$ comes as a result of the probability distributions over all actions for both $\pi_1$ and $\pi_2$ adding up to 1. That entails that, if there exists a certain action for which $\pi_2(a'|s) - \pi_1(a'|s) = \epsilon_\pi$, then there must be at least another action $a''$ for which $\pi_1(a''|s) > \pi_2(a''|s)$. Therefore, the limit case is that in which half of the actions fulfill $\pi_2(a'|s) - \pi_1(a'|s) = \epsilon_\pi$, the other half fulfill $\pi_1(a'|s) - \pi_2(a'|s) = \epsilon_\pi$, and for only one of the halves $P(r|s, a) > 0$.

Then, the bound for expression (4.14) can be computed as (using $\beta_1$ and $\beta_2$ to represent $\beta(s, r, \pi_1)$ and $\beta(s, r, \pi_2)$, respectively):

$$|\beta_1 \log \beta_1 - \beta_2 \log \beta_2| \leq |\log(\min(\beta_1, \beta_2)) + 1||\beta_1 - \beta_2| \leq \left|\log \frac{p_r}{k_a} + 1\right| \frac{1}{2}|\mathcal{A}|\epsilon_\pi \tag{4.20}$$

Next, the focus shifts to the terms from Equation (4.13) in orange. Now the

objective is to find a bound for the expression

$$\left|\beta(s,r,\pi_2)\log\left(\sum_{s'\in\mathcal{S}}P_S(s')\beta(s',r,\pi_2)\right)-\beta(s,r,\pi_1)\log\left(\sum_{s'\in\mathcal{S}}P_S(s')\beta(s',r,\pi_1)\right)\right| \tag{4.21}$$

**Assumption 5** $P_S(s)\geq 1/k_s, \forall s$, with $k_s\in\mathbb{R}^+$ and $k_s\geq|\mathcal{S}|$. Similarly to Assumption 4, it is assumed that all states have a certain probability of occurring that is greater than $1/k_s$.

Next, a similar approach is followed, using the multivariate version of the MVT. The multivariate MVT states that, given a continuous function $f:\mathbb{R}^N\to\mathbb{R}$, derivable between $\boldsymbol{a}$ and $\boldsymbol{b}$, the following holds

$$|f(\boldsymbol{a})-f(\boldsymbol{b})|\leq|\nabla f(\boldsymbol{x})||(\boldsymbol{a}-\boldsymbol{b})| \tag{4.22}$$

For the case under consideration, let $f$ be the function

$$f(\boldsymbol{x})=x_k\log\left(\sum_i^N p_ix_i\right)=x_k\log\left(p_1x_1+p_2x_2+...+p_Nx_N\right) \tag{4.23}$$

This function can be mapped to each of the two elements in expression (4.21), as $x_k$ corresponds to $\beta(s,r,\pi)$ and $\log\left(\sum_i^N p_ix_i\right)$ corresponds to $\log\left(\sum_{s'\in\mathcal{S}}P_S(s')\beta(s',r,\pi)\right)$. The partial derivatives $\nabla f(\boldsymbol{x})$ are the same for all $x_i$ except for $x_k$. For $x_k$:

$$\nabla f(\boldsymbol{x})|_k=\log\left(\sum_i^N p_ix_i\right)+\frac{p_kx_k}{\sum_i^N p_ix_i}\equiv\log\left(\sum_{s'\in\mathcal{S}}P_S(s')\beta(s',r,\pi)\right)+\frac{P_S(s)\beta(s,r,\pi)}{\sum_{s'\in\mathcal{S}}P_S(s')\beta(s',r,\pi)} \tag{4.24}$$

For any other $x_i$ with $i\neq k$:

$$\nabla f(\boldsymbol{x})|_i=\frac{p_ix_k}{\sum_i^N p_ix_i}\equiv\frac{P_S(s_i)\beta(s,r,\pi)}{\sum_{s'\in\mathcal{S}}P_S(s')\beta(s',r,\pi)},\quad\text{where } s_i\in\mathcal{S}\backslash\{s\} \tag{4.25}$$

The different variables in the function are given by the different states in $\mathcal{S}$, and the points to be compared correspond to the two policies $\pi_1$ and $\pi_2$. Therefore expression (4.22) corresponds to

$$\begin{aligned}&|f(\boldsymbol{\beta}(\boldsymbol{s},r,\pi_1))-f(\boldsymbol{\beta}(\boldsymbol{s},r,\pi_2))|\leq|\nabla f(\boldsymbol{\beta}(\boldsymbol{s},r,\pi))|\,|\boldsymbol{\beta}(\boldsymbol{s},r,\pi_1)-\boldsymbol{\beta}(\boldsymbol{s},r,\pi_2)|\leq\\&\leq\max_{\boldsymbol{\beta}(\boldsymbol{s},r,\pi)}|\nabla f(\boldsymbol{\beta}(\boldsymbol{s},r,\pi))|\,|\boldsymbol{\beta}(\boldsymbol{s},r,\pi_1)-\boldsymbol{\beta}(\boldsymbol{s},r,\pi_2)|\end{aligned} \tag{4.26}$$

From Equation (4.19), each of the components of $|\boldsymbol{\beta}(\boldsymbol{s},r,\pi_1)-\boldsymbol{\beta}(\boldsymbol{s},r,\pi_2)|$ is bounded by $|\mathcal{A}|\epsilon_\pi$. Considering the assumptions introduced so far, expression (4.26) can be

turned into

$$\max_{\boldsymbol{\beta}(\boldsymbol{s},r,\pi)} |\nabla f(\boldsymbol{\beta}(\boldsymbol{s},r,\pi))|\,|\boldsymbol{\beta}(\boldsymbol{s},r,\pi_1)-\boldsymbol{\beta}(\boldsymbol{s},r,\pi_2)| \leq \frac{1}{2}|\mathcal{A}|\epsilon_\pi \max_{\boldsymbol{\beta}(\boldsymbol{s},r,\pi)} |\nabla f(\boldsymbol{\beta}(\boldsymbol{s},r,\pi))|\,\mathbf{1} =$$

$$= \frac{1}{2}|\mathcal{A}|\epsilon_\pi \max_{\boldsymbol{\beta}(\boldsymbol{s},r,\pi)} \left(\log\left(\sum_{s'\in\mathcal{S}} P_S(s')\beta(s',r,\pi)\right) + \sum_{s''\in\mathcal{S}} \frac{P_S(s'')\beta(s,r,\pi)}{\sum_{s'\in\mathcal{S}} P_S(s')\beta(s',r,\pi)}\right) =$$

$$= \frac{1}{2}|\mathcal{A}|\epsilon_\pi \max_{\boldsymbol{\beta}(\boldsymbol{s},r,\pi)} \left(\log\left(\sum_{s'\in\mathcal{S}} P_S(s')\beta(s',r,\pi)\right) + \frac{\beta(s,r,\pi)}{\sum_{s'\in\mathcal{S}} P_S(s')\beta(s',r,\pi)}\right) \leq$$

$$\leq \frac{1}{2}|\mathcal{A}|\epsilon_\pi \left|\log\left(\frac{|\mathcal{S}|}{k_s}\frac{p_r}{k_a}\right) + \frac{\beta(s,r,\pi)}{\sum_{s'\in\mathcal{S}} \frac{1}{k_s}\beta(s',r,\pi)}\right| =$$

$$= \frac{1}{2}|\mathcal{A}|\epsilon_\pi \left|\log\left(\frac{|\mathcal{S}|}{k_s}\frac{p_r}{k_a}\right) + k_s\frac{\beta(s,r,\pi)}{\sum_{s'\in\mathcal{S}} \beta(s',r,\pi)}\right| \leq \tag{4.27}$$

$$\leq \frac{1}{2}|\mathcal{A}|\epsilon_\pi \left|\log\left(\frac{|\mathcal{S}|}{k_s}\frac{p_r}{k_a}\right) + k_s\right| \tag{4.28}$$

where $|\mathcal{S}|$ is the number of different states in the MDP. Therefore, combining expressions (4.21) and (4.28):

$$\left|\beta(s,r,\pi_2)\log\left(\sum_{s'\in\mathcal{S}} P_S(s')\beta(s',r,\pi_2)\right) - \beta(s,r,\pi_1)\log\left(\sum_{s'\in\mathcal{S}} P_S(s')\beta(s',r,\pi_1)\right)\right| \leq$$

$$\leq \frac{1}{2}|\mathcal{A}|\epsilon_\pi \left|\log\left(\frac{|\mathcal{S}|}{k_s}\frac{p_r}{k_a}\right) + k_s\right| \tag{4.29}$$

Then, going back to expression (4.13), the following can be concluded

$$\alpha(s,r) \leq \left|\log\frac{p_r}{k_a} + 1\right| \frac{1}{2}|\mathcal{A}|\epsilon_\pi + \frac{1}{2}|\mathcal{A}|\epsilon_\pi \left|\log\left(\frac{|\mathcal{S}|}{k_s}\frac{p_r}{k_a}\right) + k_s\right| =$$

$$= \frac{1}{2}|\mathcal{A}|\epsilon_\pi \left(\left|\log\frac{p_r}{k_a} + 1\right| + \left|\log\left(\frac{|\mathcal{S}|}{k_s}\frac{p_r}{k_a}\right) + k_s\right|\right) \tag{4.30}$$

Finally, the MI difference $I(S,R;\pi_1) - I(S,R;\pi_2)$ is bounded by

$$
\begin{aligned}
|I(S,R;\pi_1) - I(S,R;\pi_2)| &= \left| \sum_{s\in\mathcal{S}} P_S(s) \sum_{r\in\mathcal{R}} \alpha(s,r) \right| \leq \\
&\leq \sum_{s\in\mathcal{S}} P_S(s) \sum_{r\in\mathcal{R}} \frac{1}{2}|\mathcal{A}|\epsilon_\pi \left( \left| \log \frac{p_r}{k_a} + 1 \right| + \left| \log \left( \frac{|\mathcal{S}|}{k_s} \frac{p_r}{k_a} \right) + k_s \right| \right) = \\
&= \sum_{s\in\mathcal{S}} P_S(s) |\mathcal{R}| \frac{1}{2}|\mathcal{A}|\epsilon_\pi \left( \left| \log \frac{p_r}{k_a} + 1 \right| + \left| \log \left( \frac{|\mathcal{S}|}{k_s} \frac{p_r}{k_a} \right) + k_s \right| \right) = \\
&= \frac{1}{2}|\mathcal{R}||\mathcal{A}|\epsilon_\pi \left( \left| \log \frac{p_r}{k_a} + 1 \right| + \left| \log \left( \frac{|\mathcal{S}|}{k_s} \frac{p_r}{k_a} \right) + k_s \right| \right) \qquad (4.31)
\end{aligned}
$$

where $|\mathcal{R}|$ is the total number of different rewards.

This section concludes that, in a discrete MDP with state space $\mathcal{S}$, action space $\mathcal{A}$, and reward set $\mathcal{R}$, with $\sum_{a'\in\mathcal{A}} P(r|s,a') \geq p_r$, $\pi(a|s) \geq 1/k_a$, and $P_S(s) \geq 1/k_s$, $\forall s,a,r$ ; the difference in MI between states and rewards given by two similar policies $\pi_1$ and $\pi_2$ (i.e., $|\pi_1(a|s) - \pi_2(a|s)| \leq \epsilon_\pi$, $\forall a,s$) is bounded as

$$
|I(S,R;\pi_1) - I(S,R;\pi_2)| \leq \frac{1}{2}|\mathcal{R}||\mathcal{A}|\epsilon_\pi \left( \left| \log \frac{p_r}{k_a} + 1 \right| + \left| \log \left( \frac{|\mathcal{S}|}{k_s} \frac{p_r}{k_a} \right) + k_s \right| \right) \qquad (4.31)
$$

where $|\mathcal{S}|$ and $|\mathcal{R}|$ correspond to the total number of states and rewards, respectively.

### 4.5.2 How does mutual information change as the policy converges?

The previous section considered the case of the change in MI between two generic policies $\pi_1$ and $\pi_2$. This section focuses on different policies that are part of the same learning process $\pi_0, \pi_1, \pi_2, ..., \pi_N$, where $\pi_N$ is the policy when learning converges. Similarly to the previous section, we assume that the change between consecutive policies is small, i.e., $|\pi_n(a|s) - \pi_{n-1}(a|s)| \leq \lambda^{n-1}\epsilon_\pi, \forall a,s$, with $0 < \lambda < 1$. This indicates that this change becomes smaller as the policy converges.

The maximum change between policy $\pi_n$ and policy $\pi_0$ is bounded as follows

$$|\pi_n(a|s) - \pi_0(a|s)| \leq \sum_{k=1}^{n} \lambda^{k-1}\epsilon_\pi = \epsilon_\pi \sum_{b=0}^{n-1} \lambda^b = \epsilon_\pi \left( \sum_{b=0}^{\infty} \lambda^b - \sum_{n}^{\infty} \lambda^b \right) = \epsilon_\pi \frac{1-\lambda^n}{1-\lambda} \tag{4.32}$$

Then, a similar derivation to Section 4.5.1's with a change in inequality (4.19) is followed

$$\beta(s, r, \pi_n) - \beta(s, r, \pi_0) = \sum_{a \in \mathcal{A}} P(r|s, a)[\pi_n(a|s) - \pi_0(a|s)] \leq \frac{1}{2}|\mathcal{A}|\epsilon_\pi \frac{1-\lambda^n}{1-\lambda} \tag{4.33}$$

Now, by taking into account this change into expressions (4.20) and (4.28), one can conclude that the difference in MI as the policy converges is bounded by

$$|I(S, R; \pi_n) - I(S, R; \pi_0)| \leq \frac{1}{2}|\mathcal{R}||\mathcal{A}|\epsilon_\pi \frac{1-\lambda^n}{1-\lambda} \left( \left| \log \frac{p_r}{k_a} + 1 \right| + \left| \log \left( \frac{|\mathcal{S}|}{k_s} \frac{p_r}{k_a} \right) + k_s \right| \right) \tag{4.34}$$

### 4.5.3 The effect of removing features

Finally, this part considers the case in which one or more features from the state space are removed and the agent is retrained from scratch. In this case, the updated space is defined as $\mathcal{S}'$, with $|\mathcal{S}'| \leq |\mathcal{S}|$. Note the number of different states cannot increase, see the example in Figure 4-4.

As discussed in Section 4.4.2, the maximum MI that can be achieved corresponds to $\log N_S$, where $N_S$ is the number of states in the MDP. Thus, when a feature is removed from the state space, the maximum possible MI also decreases since there is less information available to encode.

However, the bound derived in Equation (4.34) remains valid in this case with an updated coefficient $k_{s'}$. By definition, $k_{s'} \leq k_s$, where $k_s$ is the coefficient associated with the original state space $\mathcal{S}$. Consequently, the bound on the change in MI becomes:

$$|I(S', R; \pi_n) - I(S', R; \pi_0)| \leq \frac{1}{2}|\mathcal{R}||\mathcal{A}|\epsilon_\pi \frac{1-\lambda^n}{1-\lambda} \left( \left| \log \frac{p_r}{k_a} + 1 \right| + \left| \log \left( \frac{|\mathcal{S}'|}{k_{s'}} \frac{p_r}{k_a} \right) + k_{s'} \right| \right) \tag{4.35}$$

where the changes with respect to expression (4.34) are highlighted in blue.

The derived bound consists of two components. The first component,

$$\frac{1}{2}|\mathcal{R}||\mathcal{A}|\epsilon_\pi \frac{1-\lambda^n}{1-\lambda}\left(\left|\log\frac{p_r}{k_a}+1\right|\right)$$

remains constant and independent of the state space. It indicates that the reward structure and the action space also influence the MI bound. Smaller action spaces and less diversity in rewards lead to tighter MI bounds.

The second component depends on the state space and therefore is updated given the change of $\mathcal{S}$ to $\mathcal{S}'$, i.e.,

$$\frac{1}{2}|\mathcal{R}||\mathcal{A}|\epsilon_\pi \frac{1-\lambda^n}{1-\lambda}\left(\left|\log\left(\frac{|\mathcal{S}'|}{k_{s'}}\frac{p_r}{k_a}\right)+k_{s'}\right|\right)$$

The relationship between the two components determines whether the MI bound becomes looser or tighter when a feature is removed. Specifically, these expressions are compared:

$$\left|\log\left(\frac{|\mathcal{S}|}{k_s}\frac{p_r}{k_a}\right)+k_s\right| \lesseqgtr \left|\log\left(\frac{|\mathcal{S}'|}{k_{s'}}\frac{p_r}{k_a}\right)+k_{s'}\right|$$

In the case one assumes that $k_s = |\mathcal{S}|$ and $k_{s'} = |\mathcal{S}'|$, then the previous inequality evaluates whether $k_s$ is greater than $k_{s'}$, assuming $p_r$ and $k_a$ remain the same for both MDPs. While $k_s$ and $k_{s'}$ can both take arbitrarily large values, if all states are equiprobable, $k_{s'}$ will take lower values than $k_s$. Therefore:

$$\left|\log\left(\frac{|\mathcal{S}|}{k_s}\frac{p_r}{k_a}\right)+k_s\right| \geq \left|\log\left(\frac{|\mathcal{S}'|}{k_{s'}}\frac{p_r}{k_a}\right)+k_{s'}\right|$$

This implies that the MI bound becomes *tighter* when a feature is removed.

There is another case in which, after removing one feature from the state space, the MDP ends up with less possible states, i.e., $|\mathcal{S}'| < |\mathcal{S}|$, but the following holds: $k_s = k_{s'}$. For example, if there exist $s_1 \in \mathcal{S}$ and $s_2 \in \mathcal{S}'$ such that $P_{\mathcal{S}}(s_1) = P_{\mathcal{S}'}(s_2)$, and both correspond to the minimum nonzero probabilities of their respective distributions, then the value $\log(|\mathcal{S}|/k_s)$ will be greater than $\log(|\mathcal{S}'|/k_{s'})$. However, since $k_s \geq |\mathcal{S}|$, $k_a \geq 1$, and $p_r \leq 1$, then the logarithms will have negative value and the effect of taking into account the absolute value will be

$$\left|\log\left(\frac{|\mathcal{S}|}{k_s}\frac{p_r}{k_a}\right)+k_s\right| \leq \left|\log\left(\frac{|\mathcal{S}'|}{k_{s'}}\frac{p_r}{k_a}\right)+k_{s'}\right|$$

This indicates that the MI bound becomes *looser* after removing the feature.

In summary, the effect of removing features on the MI bound depends on the relationship between $k_s$ and $k_{s'}$. When $k_{s'}$ takes lower values than $k_s$, the MI bound becomes tighter. On the other hand, if $k_s = k_{s'}$ and $|\mathcal{S}'| < |\mathcal{S}|$, the MI bound becomes looser.

## 4.6 Case study: Traffic Signal Control

To ground the analysis and understanding of feature selection in RL, this section reintroduces the TSC problem presented in Section 4.2.3 as a real-world use case to study the change in MI during learning. The objective in this problem is to control the traffic lights at an intersection by setting them to different phases. A phase is defined as an assignment of states to each traffic light (e.g., green, red). The ultimate goal is to optimize the traffic flow at the intersection accounting for figures of merit such as the average waiting time. TSC problems can be formulated in multiple ways that affect the specific available actions, the timescale, and the concrete goals to achieve [106]. This dissertation focuses on the most frequent formulation, in which time is discretized in equal intervals and a phase from a set of predefined phases is chosen at each timestep. This can be applied to individual intersections or to large road networks with multiple intersections.

### 4.6.1 Evaluating mutual information between state features and rewards during learning

Several experiments are conducted to gain insights on the significance of the features in Table 4.1. To accomplish this, the RESCO benchmark for TSC [20] and a real-world road network and simulation with 21 agent-controlled traffic lights are utilized (see Figure 4-6) as the scenario. The following elements are considered in the experiments:

- **State space:** The state space used in one of the benchmark models in RESCO is taken. It consists of 5 different features for each lane in the intersection: the current phase, the number of vehicles approaching, the average waiting time, the number of vehicles waiting, and the average vehicle speed in the lane.

- **Action:** At each timestep, the action involves setting the phase at each intersection from a set of predefined phases. Each intersection may have a different number of phases.

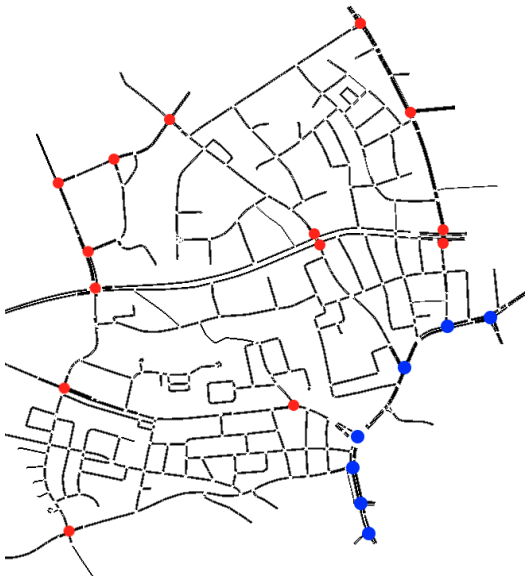


**Figure 4-6:** Road network used during the simulations, based on the town of *Ingolstadt*, in Germany. Each of the 21 dots represents an intersection controlled by an RL agent. The 7 intersections marked in blue are used as a downsized version of the environment.

- **Reward:** At a given timestep, the sum of the time each vehicle at the intersection has been waiting is computed and multiplied by -1 to obtain the reward.

- **RL algorithm:** An independent agent is set up for each of the 21 intersections, and the IDQN model from RESCO is used to train each agent independently.

During training, the states visited and the reward obtained after each action are tracked. Since MI can only be estimated based on data samples instead of distributions [247], the MI is computed using entropy estimation from k-nearest neighbors distances [219, 220, 323] for each training epoch. Additionally, a downsized version of the environment with only 7 agent-controlled intersections is considered (see Figure 4-6). More implementation details can be found in Appendix A.1.2.

Figures 4-7 and 4-8 show how the MI between each of the 5 features and the reward evolves during 100 episodes of training in the 7-intersection and 21-intersection use cases, respectively. One can observe the MI substantially changes during learning for two of the features and remains at a constant level for the rest. This is consistent with the bounds given by expressions (4.31) and (4.34), as they allow big changes in the MI but they do not necessarily imply changes in the MI, as there are circumstances in which the MI does not change, as was discussed in Section 4.5. Additionally, the figures show that changes in MI are more pronounced at the beginning of training,

which is consistent with the reasoning in Section 4.5.2.

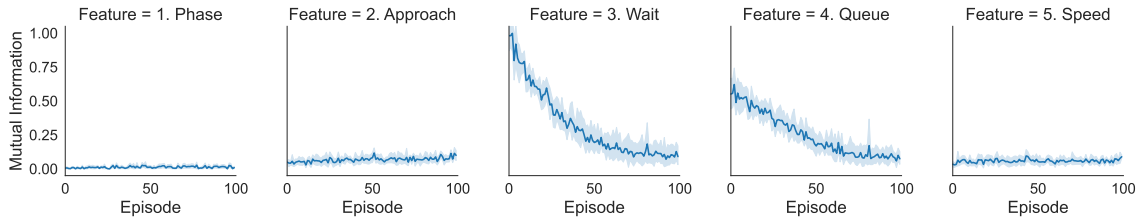



**Figure 4-7:** Evolution of the mutual information between the reward and each of the five features considered in the RESCO benchmark for Traffic Signal Control during learning in the 7-intersection use case. The mean and 95% CI across 7 intersections are shown.


**Figure 4-8:** Evolution of the mutual information between the reward and each of the five features considered in the RESCO benchmark for Traffic Signal Control during learning in the 21-intersection use case. The mean and 95% CI across 21 intersections are shown.

### 4.6.2 Which features are relevant?

Next, the relationship between the MI and the relevant features in the experiments is investigated. To that end, this section introduces a figure of merit and the use of the feature masks presented in Section 4.3. The figure of merit considered consists of the average waited time per car in the simulation for the resulting policy. Taking the average across vehicles provides a measure of the overall performance across the network, as one vehicle might cross multiple independent intersections.

Regarding the feature masks, similar to the experiments in Section 4.2.2, using feature masks entail using alternative feature spaces that zero out one or more of the five features considered in this experiment. In total, 31 different masks are used ($2^5 - 1$), which correspond to all possible combinations from one to five features, excluding the option that all features are masked. For each of the 31 masks, 5 training runs for different random seeds are used. Consequently, for any of the five

features, there are $16 \cdot 5 = 80$ figure of merit values corresponding to experiments that do use such feature, and $15 \cdot 5 = 75$ figure of merit values corresponding to experiments that do *not* use the feature. Therefore, a hypothesis test [191] between the two sets of samples for each feature can be run in order to determine whether there is a statistically significant difference between the averages of the figure of merit values for both samples. The $P$-values for each feature are reported in Table 4.2.

**Table 4.2:** P-values obtained for five different hypothesis tests comparing two sets of samples containing figure of merit values from runs in the 7-intersection case using each feature and not using it, respectively.

| 1. Phase | 2. Approach | 3. Wait | 4. Queue | 5. Speed |
|---|---|---|---|---|
| 0.044 | 0.799 | $< 0.001$ | $< 0.001$ | 0.479 |

The resulting values show that the waiting time and the queue size are strongly relevant features, in addition to the current phase (i.e., a $P$-value of 0.05 is used to determine statistical significance). In contrast, the number of cars approaching and the average speed of the cars are irrelevant features for this problem. It is interesting to point out that in Figures 4-7 and 4-8, high MI during learning for the two strongly relevant features is observed, but it diminishes as the policy converges, showing a similar MI to that of the irrelevant features. Therefore, in this case, relying on the MI to detect relevant features will be more or less effective depending on the moment of the learning process it is being measured at.

Another interesting observation is that, although the MI might not be robust to policy changes, it is a cheaper method to evaluate feature relevance. While there is a correspondence between the results in Figure 4-7 and in Table 4.2, the former require a single training run whereas the latter require multiple different training runs, which might be impractical if the number of features is large. Therefore, there is a cost-efficiency advantage in understanding the evolution of MI under policy changes in order to identify relevant features. Appendix E.2 shows that similar conclusions can be derived when looking at the MI between pairs of features as the policy changes.

## 4.7 Chapter summary and contributions

This chapter has expanded on real-world RL design by focusing on a particular component, the state space, and providing a comprehensive analysis on how to improve current state space design practices based on mutual information (MI) analyses. This

is motivated by two research opportunities identified in the literature review: a prevailing lack of design consensus and a strong reliance on human-driven design in domain-specific RL research. The goal of this chapter has been to study how feature-based state spaces are currently designed and to propose a practical approach to enhance these processes. Improving state space design practices can potentially lead to less intrusive and less costly data collection needs.

The chapter has begun by examining the related literature review, identifying the MI as one popular method of carrying out feature selection in RL and other areas of Machine Learning. Then, I have presented two important issues of current state space design strategies: using an excess of features and a lack of consensus about which feature sets to use in specific applications. Then, I have formalized the problem of feature selection in RL, with the goal of identifying the best feature set for a given problem.

The next part of the chapter has focused on MI. There are two main shortcomings related to MI use in RL: first, there is no reporting on MI being regularly used in domain-specific RL studies as part of the design process; state spaces are treated simply as elements to be reported. Second, domain-agnostic studies on MI for feature selection in RL do not consider scenarios in which the policy changes as the agent learns; policies are mostly treated as static. To emphasize the role of the policy when looking at MI, I have presented a toy example in which, by modifying the agent's policy, the observed MI between certain features and the reward can be driven to both the minimum and maximum possible values, which renders such features useless and indispensable, respectively.

To further understand the behavior of MI under policy changes, the chapter has continued by deriving a mathematical bound on how much the MI can change between two different policies for the same Markov Decision Process. Then, the specific case of a policy changing until convergence has been considered and the aforementioned bound has been updated, concluding that the MI may suffer substantial changes as the policy converges, although these changes become smaller the closer we are from the convergence point. Finally, I have presented an updated bound that results when removing a feature or set of features from the state space. This updated bound can be tighter or looser depending on the structure of the updated state space.

Lastly, this chapter has concluded by taking the Traffic Signal Control (TSC) problem as a real-world use case in which to validate the observations around MI. By relying on the RESCO benchmark, we have observed that 1) using all features is not the best approach, 2) the MI might be a good and cost-efficient way of identifying

useful features in high-dimensional contexts, and 3) the MI of relevant and non-relevant features become very similar once the policy converges, which emphasizes the need of observing the MI along the complete learning curve.

The contributions of this chaper are the following:

**Contribution 4.1** Characterized the limitations of current state space design practices for real-world RL problems, identifying an excess of features and a lack of consensus as two important shortcomings.

**Contribution 4.2** Identified a lack of study of feature selection via mutual information in RL in the context of a changing policy and motivated its consideration by proving the policy can substantially influence the observed mutual information.

**Contribution 4.3** Derived a mathematical bound for how much the mutual information between state features and rewards can change when considering two different policies acting on the same Markov Decision Process.

**Contribution 4.4** Derived a mathematical bound for how much the mutual information between state features and rewards can change as the policy converges in the context of an RL algorithm.

**Contribution 4.5** Outlined the possible updates on mutual information bounds when removing a set of features from the state space.

**Contribution 4.6** Provided a better feature set for two use cases of the RESCO benchmark for Traffic Signal Control. Demonstrated this feature set is also identifiable via mutual information.

**Contribution 4.7** Validated the observations of mutual information changes during policy learning using a real-world application and motivated the need to compute the mutual information at different points of the convergence process to get reliable estimations of feature relevance.

# Chapter 5

# MetaPG: Optimizing Multiple Reinforcement Learning Objectives

This chapter introduces MetaPG, a method to evolve a population of actor-critic RL algorithms, identified by their loss functions, with the goal of optimizing multiple RL objectives. Section **5.1** introduces the goal of the method framed as a way of designing RL algorithms while accounting for multiple real-world challenges. Then, Section **5.2** describes the main components of MetaPG and its workflow. The remainder of the chapter focuses on detailed descriptions of each of these components. Section **5.3** explains the representation procedure based on computational graphs. Next, the main building blocks of the evolution mechanism are introduced in Section **5.4**. Following, Section **5.5** explains the encoding of RL objectives as fitness scores into MetaPG. Then, Section **5.6** describes how to run experiments with MetaPG, including an overview of the meta-training, meta-validation, and meta-testing phases. Finally, the summary of the chapters and its contributions are presented in Section **5.7**.

## 5.1 Introduction

In Chapter 1, I introduced the so-called challenges of real-world RL as specific issues that manifest in numerous practical applications and contribute to a lack of robustness in RL. In Chapter 2, I reviewed the literature around those challenges and identified that, while in real-world problems they manifest themselves in combination, there is little research on addressing multiple challenges at the same time. The combined effect of real-world challenges can be exponentially detrimental even for state-of-the-art RL algorithms [100]. The majority of studies proposing new algorithms to robustify

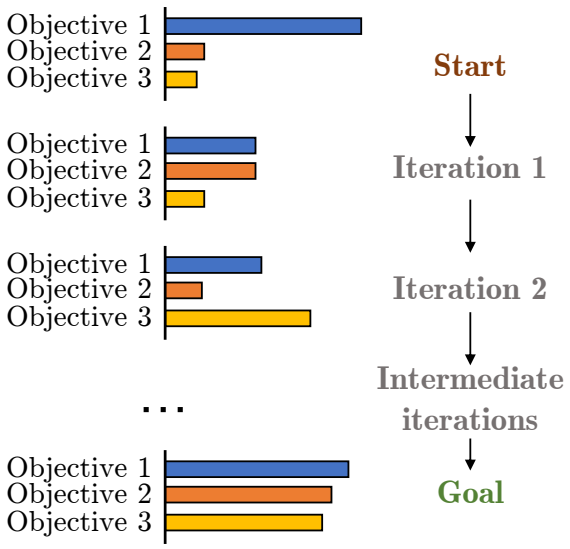



**Figure 5-1:** In many practical contexts, designing the RL agent that fulfills multiple objectives above a certain threshold is an empirical and costly iteration-based process.

RL against individual challenges have a single-objective focus and come from human-driven design processes. These design practices might become prohibitively expensive when trying to optimize more than one RL objective, especially in concrete real-world problems, where design tends to be empirical [16,234]. In order to find new algorithms that are robust against multiple challenges given by different objectives, the design process might require numerous expensive iterations, as trade-offs among objectives might be present (see Figure 5-1). This motivates looking into AutoML methods for RL, i.e., AutoRL, as a way to introduce automaton into the design process and make it more cost-efficient.

This chapter tackles these gaps and proposes MetaPG[1], a method that relies on AutoML to evolve a population of RL algorithms, identified by their loss functions, with the goal of optimizing multiple RL objectives. MetaPG, which I initially published in [134], relies on a symbolic search language to represent algorithms as graphs, on independent fitness scores, and on multi-objective optimization routines to find new algorithms that are aligned with the objectives considered. This method aims to find algorithm improvement directions that jointly optimize all objectives until it obtains a Pareto Front or Pareto-optimal set of loss functions that maximizes fitness with respect to each objective, approximating the underlying trade-off between them.

[1]The name MetaPG comes from Meta Policy Gradient. MetaPG specifically evolves actor-critic loss functions.

MetaPG is of interest to both domain-agnostic and domain-specific RL research. On one hand, it highlights an important aspect of designing RL algorithms for practical applications: the balancing of multiple real-world challenges that together pose bottlenecks for deployment. On the other hand, it provides practitioners without RL expertise with a way of automating the search process. This way, practitioners only have to limit themselves to encode their multiple problem needs as independent fitness scores. I believe that MetaPG can benefit AutoML research too, as automating RL still remains a complex research problem due to its brittleness and complexity [296].

This chapter describes MetaPG at a methods level, detailing each of its components. Then, Chapter 6 presents the application of MetaPG to the specific use case of optimizing performance, generalizability, and stability.

### 5.1.1 Related Work

The automation of the design process of RL components via AutoRL has been explored in different directions [296], as the design process (see RL roadmap in Figure 3-3) include multiple elements that can be automated. There have been studies focused on automating the search for RL algorithms [29, 67, 212, 280], others have focused on hyperparameter optimization in RL [165, 395, 404], the policy or neural network has also been the subject of study in other works [129, 266], and finally, there has been research on automating environmental components such as the reward function [89, 110, 112, 120, 149, 371]. This chapter specifically addresses evolving RL loss functions and leaves other elements of the RL system out of the scope.

MetaPG relies on evolution to search over the space of loss functions. In the context of AutoML, these were introduced by neuro-evolution [267, 343]; one of their major contributions has been in the field of neural network architecture search [199, 313, 342]. Regarding RL, evolution has been proposed as a method to search for policy gradients [176] and value learning losses [67]. The work in this chapter is also related to the field of genetic programming, in which the goal is to discover computer code [67, 218, 315]. MetaPG uses a multi-objective evolutionary method, NSGA-II [85], to discover new RL algorithms, specifically actor-critic algorithms [348], represented as graphs that do not have meta-parameters to be learned.

Finally, the search over loss function spaces has already been explored in the literature. One popular approach is to use neural loss functions whose parameters are optimized via meta-gradient [29, 212, 244, 280, 394]. An alternative is to use symbolic representations of loss functions and formulate the problem as optimizing over a

combinatorial space. One example is [15], which represents extrinsic rewards as a graph and optimizes it by cleverly pruning a search space. Learning value-based RL loss functions by means of evolution was first proposed by Co-Reyes et al. [67], and was applied to solving discrete action problems. He et al. [161] propose a method to evolve auxiliary loss functions which complemented predefined standard RL loss functions. MetaPG focuses on continuous control problems and searches for complete symbolic loss functions of actor-critic algorithms.

## 5.2 Method Overview

MetaPG (see Figure 5-2) focuses on finding new actor-critic RL algorithms given by their loss functions (policy loss and critic loss). To that end, it represents loss functions as directed acyclic graphs using a symbolic language. Multiple graphs form a population that is constantly evolved by mutating individual graphs. The population can be initialized from scratch (using randomly-generated graphs) or using predefined graphs as warm-starts. To determine the best graphs in the population, MetaPG relies on a ranking mechanism that takes into account different independent fitness scores in order to determine the best graphs in the population. The fitness scores are measured by training an RL agent from scratch with the corresponding loss function in one or more training environments and evaluating aspects related to the metric of interest. The multi-objective evolutionary algorithm NSGA-II [85] is used to jointly optimize all fitness scores until growing a Pareto-optimal set of graphs or Pareto Front. Finally, by inspecting the graphs of the evolved algorithms, MetaPG allows to interpret which substructure drive the gains for the particular objectives considered.

Algorithm 1 summarizes the overall process. Given a set of training environments $\mathcal{E}$ and an evaluation subroutine `Eval`, a population of graphs $P_0$ is initialized from scratch or using a warm-start. Before beginning evolution, this population is evaluated using the subroutine Eval$(L, \mathcal{E})$, which returns an array of fitness scores, one for each independent objective considered. Then, using NSGA-II's `Offspring` subroutine, a new set of evolved graphs $Q_0$ is evolved from $P_0$ and then evaluated using `Eval`. This process is repeated multiple times. To keep the population at a constant size every time an offspring is generated, the function `RankAndSelect` is utilized to pick the best graphs; this is given by NSGA-II too. After completing $G$ iterations, the algorithm returns the Pareto Front of the final population.

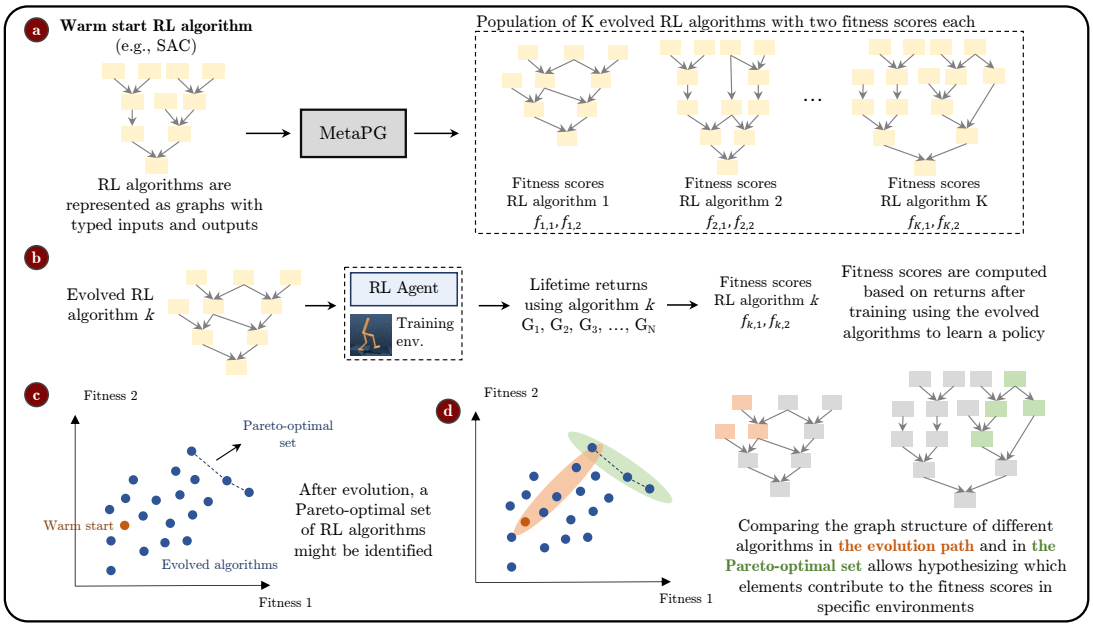



**Figure 5-2:** MetaPG overview, example with two fitness scores encoding two RL objectives. **(a)** The method starts by taking a warm-start RL algorithm with its loss function represented in the form of a directed acyclic graph. MetaPG consists of a meta-evolution process that, after initializing algorithms to the warm-start, discovers a population of new algorithms. **(b)** Each evolved graph is evaluated by training an agent following the algorithm encoded by it, and then computing two fitness scores based on the training outcome. **(c)** After evolution, all RL algorithms can be represented in the fitness space and a Pareto-optimal set of algorithms can be identified. **(d)** Identifying which graph substructures change across the algorithms in the Pareto set reveals which operations are useful for specific RL objectives. MetaPG can be scaled to more than two RL objectives.

## 5.3 RL algorithm representation

MetaPG encodes loss functions as graphs consisting of typed nodes sufficient to represent a wide class of actor-critic algorithms. Compared to the prior value-based RL evolutionary search method introduced by [67], MetaPG's search space greatly expands on it and adds input and output types to manage the search complexity. Nodes in the graph encode loss function inputs, operations, and loss function outputs. The inputs include elements from transition tuples coming from a replay buffer, constants such as the discount factor $\gamma$, a policy network $\pi$, and multiple critic networks $Q_i$. Operation nodes support intermediate algorithm instructions such as basic arithmetic neural network operations. Then, outputs of the graphs correspond to the policy and critic losses. The gradient descent minimization process takes these out-

**Algorithm 1** MetaPG Overview

**Input:** Training environments $\mathcal{E}$, `Eval` subroutine
**Initialize**: Initialize population $P_0$ of loss function graphs (random initialization or bootstrap with an algorithm such as SAC).

**for** $L$ in $P_0$ **do** $L.score \leftarrow \texttt{Eval}(L, \mathcal{E})$
**end for**
$Q_0 \leftarrow \texttt{Offspring}(P_0)$ ▷ NSGA-II
**for** $L$ in $Q_0$ **do** $L.score \leftarrow \texttt{Eval}(L, \mathcal{E})$
**end for**
**for** $t = 1$ **to** $G$ **do**
    $R \leftarrow P_{t-1} \bigcup Q_{t-1}$
    $P_t \leftarrow \texttt{RankAndSelect}(\text{R})$ ▷ NSGA-II
    $Q_t \leftarrow \texttt{Offspring}(P_t)$ ▷ NSGA-II
    **for** $L$ in $Q_t$ **do** $L.score \leftarrow \texttt{Eval}(L, \mathcal{E})$
    **end for**
**end for**
**Output:** Pareto Front of all loss function graphs.

puts and computes their gradient with respect to the respective network parameters. The experimental evaluation in Chapter 6 only considers steepest gradient descent to update network parameters; incorporating other gradient descent strategies into the search space out of the scope of this dissertation. However, other strategies such as natural gradient [254] or conjugate gradient [332] could be incorporated, as they do gradient transformations and are agnostic to loss functions.

### 5.3.1 Representing loss functions as graphs

Figure 5-3 shows how a simple RL algorithm, REINFORCE, is turned into a computational graph. Variables in the loss function correspond to input nodes (blue) and operations in the loss function expression conform the set of intermediate nodes (orange). In this case, the parameter update is used as the output of the graph.

### 5.3.2 Search space

MetaPG admits both continuous and discrete action spaces; specific nodes in the graphs —e.g., the neural networks— are adapted to work with the corresponding space. The majority of operation nodes treat input elements as tensors with variable shapes in order to maximize graph flexibility. Each node possesses a certain number

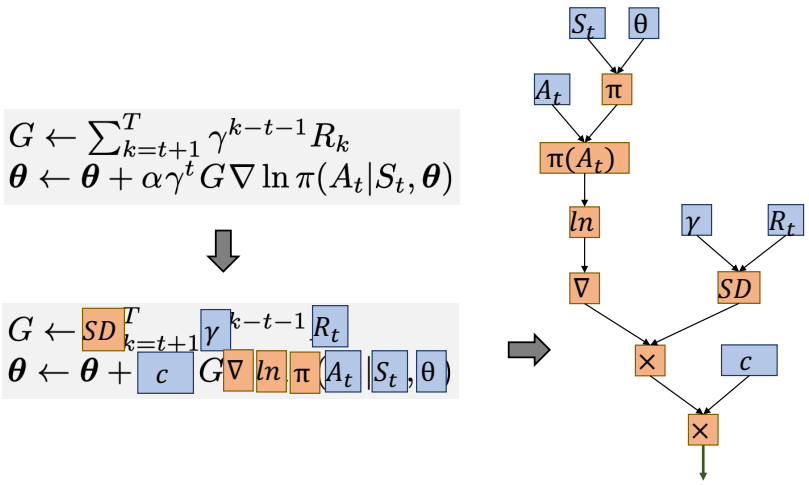



**Figure 5-3:** Using REINFORCE [348] as an example to show the transformation of loss functions into computational graphs.

of input and output edges, which are determined by the specific operation this node carries out. For example, a node that takes in two tensors and multiplies them element-wise has two input edges and a single output edge. MetaPG's search language supports both on-policy and off-policy algorithms; however, this dissertation focuses on off-policy algorithms given their better sample efficiency.

The complete list of the nodes considered is the following:

**Input nodes** MetaPG only encodes canonical RL elements as inputs:

- Policy network $\pi$
- Two critic networks $Q_{1,2}$ and two target critic networks $Q_{targ_{1,2}}$
- Batch of states $s_t$ and next states $s_{t+1}$
- Batch of actions $a_t$
- Batch of rewards $r_t$
- Discount factor $\gamma$

**Output nodes** The output of these nodes is used as loss function to compute gradient descent on:

- Policy loss $L_\pi$
- Critic loss $L_{Q_i}$

**Operation nodes** These nodes operate generally on tensors and can broadcast operations when input sizes do not match:

- Addition: add two, three, or four tensors
- Multiplication: compute element-wise product of two or three tensors
- Subtract two tensors
- Divide two tensors and add constant $\epsilon$ to the denominator
- Neural network operation: Action distribtuion from state
- Neural network operation: stopping gradient computation
- Operations with action distributions: Sample, Log-probability
- Mean, sum, and standard deviation over last axis of array or over entire array
- Cumulative sum, cumulative sum with discount
- Squared difference
- Multiply by a constant: -1, 0.1, 0.01, 0.5, 2.0
- Minimum and maximum over last axis of a tensor
- Minimum and maximum element-wise between two tensors
- Other general operations: clamp, absolute value, square, logarithm, exponential
- Trigonometry functions

To get an upper bound of the size of the search space in terms of the number of possible graphs (not all of them valid), let's consider the $(k+1)$-th node in the graph of size $K$ nodes. This node can correspond to one of the $N$ different operation nodes. Assuming that this node has two inputs, there are $\binom{k}{2}$ possibilities of connecting to previous nodes in the graph. Therefore, there are a total of $\frac{1}{2}Nk(k-1)$ possible combinations for the $(k+1)$-th node. Then, an upper bound of the total number of possible graphs is[2]

$$\left(\frac{NK(K-1)}{2}\right)^K \tag{5.1}$$

In the experiments in Chapter 6, the number of nodes per graph is limited to 60 and 80. With $N = 33$ and $K = 60$, the search space has approximately $10^{286}$ graphs. This number increases to $10^{401}$ if $K = 80$ instead.

### 5.3.3 Example algorithm: Soft Actor-Critic (SAC)

This section presents an example of an RL algorithm represented using MetaPG's symbolic graph language. Specifically, Figure 5-4 depicts the version of Soft Actor-Critic (SAC) [152] used in this dissertation[3]. The equations for the policy loss $L_\pi^{WS}$

[2]Equation 5.1 assumes the operator is commutative. For non-commutative operations the 1/2 factor does not apply.

[3]The specific names shown in the picture correspond to function names encoding each operation in the code

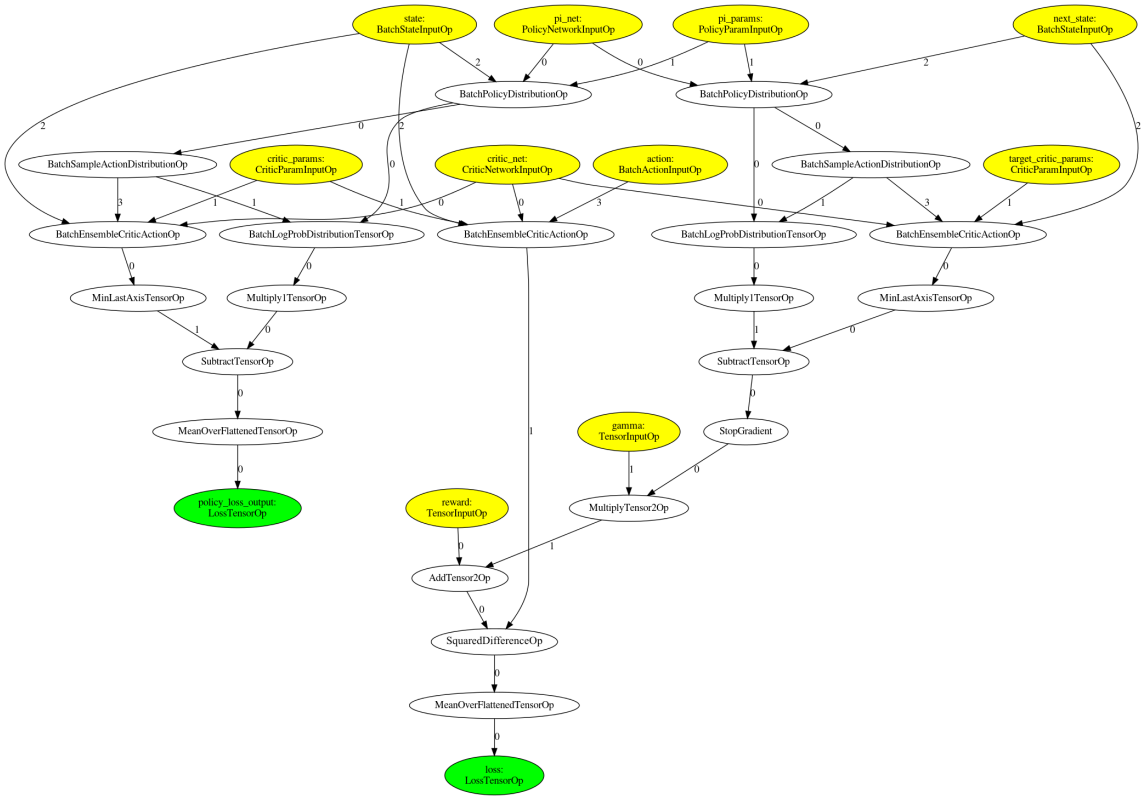



**Figure 5-4:** Soft Actor-Critic (SAC) algorithm represented as a graph using MetaPG's symbolic language.

and critic loss $L_{Q_i}^{WS}$ correspond to:

$$L_{\pi}^{WS} = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\log\pi(\tilde{a}_t|s_t) - \min_i Q_i(s_t,\tilde{a}_t)\right] \tag{5.2}$$

$$L_{Q_i}^{WS} = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left(r_t + \gamma\left(\min_i Q_{targ_i}(s_{t+1},\tilde{a}_{t+1}) - \log\pi(\tilde{a}_{t+1}|s_{t+1})\right) - Q_i(s_t,a_t)\right)^2\right] \tag{5.3}$$

where $\tilde{a}_t \sim \pi(\cdot|s_t)$, $\tilde{a}_{t+1} \sim \pi(\cdot|s_{t+1})$, and $\mathcal{D}$ is a dataset from the replay buffer. Graph representations for other relevant RL algorithms are presented in Appendix B.

## 5.4 Evolution details

This section presents an overview of the different components of the evolution module used in MetaPG. Specific implementation aspects related to the experimental setups are discussed in Chapter 6.

**Mutation** The population can be initialized from scratch or using a warm-start RL algorithm; all individuals are copies of this algorithm's graph at the beginning.

Once the population is initialized, individuals undergo mutations that change the structure of their respective graphs. Specifically, mutations consist of either replacing one or more nodes in the graph or changing the connectivity for one edge. The specific number of nodes that are affected by mutation is randomly sampled for each individual.

**Operation consistency** To prevent introducing corrupted child graphs into the population during the mutation process, MetaPG checks operation consistency, i.e., for each operation, it makes sure the shapes of the input tensors are valid and compatible, and computes the shape of the output tensor. These shapes and checks are propagated along the computational graph.

**Hashing** To avoid repeated evaluations, MetaPG hashes [315] all graphs in the population. Once the method produces a child graph and proves its consistency, it computes a hash value and, in case of a cache hit, reads the fitness scores from the cache. Since only the gradient of a loss function matters during training, MetaPG hashes a graph by computing the corresponding loss function's gradient on synthetic inputs. In this case, not only preventing the evaluation of the same graph twice is important, but also identifying graphs that are different in form but identical in function (e.g., adding a node that multiplies by 1 affects the form but not the function of the graph). To that end, before hashing MetaPG prunes all graphs so that only nodes that contribute to the output are taken into account. Then, it looks at the gradients of the output losses with respect to synthethic input parameters and uses their concatenation as the hash value.

**Fitness scores and Pareto-dominance** MetaPG keeps the population to a fixed size during evolution. To decide which individuals should be removed in the process, the method makes use of different fitness scores that encode each of the RL objectives considered. These scores are not combined but treated separately in a multi-objective fashion. This means that, after evaluating a graph $i$, it will have fitness scores $\{f_{i,1}, f_{i,2}, ..., f_{i,F}\}$, where $F$ is the number of objectives considered. Then, when comparing two graphs $i$ and $j$, we say $i$ has higher fitness than $j$ if and only if $f_{i,k} \geq f_{j,k}, \forall k$, with at least one fitness score $k'$ such that $f_{i,k'} > f_{j,k'}$. In this case we also say graph $i$ Pareto-dominates graph $j$. If neither $i$ Pareto-dominates $j$ nor vice versa, we say both graphs are Pareto-optimal (e.g., if $f_{i,k'} > f_{j,k'}$ and $f_{i,k''} < f_{j,k''}$, then $i$ and $j$ are Pareto-optimal).

**Pruning the population** The process of removing individuals from the population follows the NSGA-II algorithm [85] which, assuming a maximum population size of $P_{max}$ individuals:

1. From a set of $P$ individuals, with $|P| > P_{max}$, it computes the set $P_{opt}$ of Pareto-optimal fittest graphs. None of the graphs in $P_{opt}$ is Pareto-dominated by any other graph in the population and, if a graph $i$ in $P$ is Pareto-dominated by at least one other graph, then $i$ does not belong to the Pareto-optimal set.

2. If $|P_{opt}| \geq P_{max}$, the graphs are ranked based on their *crowding distance* in the fitness space. This favors individuals that are further apart from other individuals in the fitness space. The fittest $P_{max}$ individuals of the Pareto-optimal set $P_{opt}$ are kept in the population.

3. Otherwise, if $|P_{opt}| < P_{max}$, the set $P_{opt}$ is kept in the population and the process is repeated taking $P \leftarrow P \setminus P_{opt}$ and $P_{max} \leftarrow P_{max} - |P_{opt}|$.

**Hurdle evaluations** MetaPG carries out evaluations for different individuals in the population in parallel. Each evaluation might require multiple agents to be trained in different environments. To prevent spending too many resources on algorithms that are likely to yield bad policies, MetaPG uses a simple hurdle environment [67]. It first evaluates the algorithm on the hurdle environment and only proceeds with more complex and computationally expensive environments if the resulting policy performs above a certain threshold on the hurdle environment.

## 5.5 Defining fitness scores

This section delves into the design of fitness scores for MetaPG experiments, emphasizing their significance in guiding the evolutionary search process for discovering new RL algorithms. When a new algorithm is proposed, all fitness scores are computed to rank it against other individuals in the population. MetaPG is a multi-objective optimization method, allowing for the encoding of multiple fitness scores without any limit on the number of independent scores considered. Consequently, the evolutionary processes described in Section 5.4 can scale to accommodate numerous scores.

The fitness scores serve as inputs to the method, necessitating their prior design. The design process should primarily focus on capturing the system-level objectives described in Chapter 3. Therefore, each score should encode a specific aspect of the

desired agent competence. In the context of this dissertation, this presents an opportunity to not only incorporate metrics related to performance but also design fitness scores that capture metrics connected to robustness against real-world challenges.

The scoring functions are not constrained, except for the requirement to return a single scalar for each objective. As a result, algorithms can be evaluated in multiple environments or configurations, if necessary, and then aggregated to obtain the fitness score. For instance, the work in [67] employs each algorithm to train agents in various environments, considering the average of the normalized returns across these environments as the fitness score. While evaluating algorithms across multiple environments can assess their cross-environment generalization capabilities, it also increases the overall runtime of MetaPG due to the additional scoring runtime.

The number of RL objectives to optimize for does not necessarily need to correspond to the number of fitness scores. For example, in Chapter 6, MetaPG is used to optimize single-task performance, zero-shot generalizability, and stability across independent runs. These three objectives are encoded as two fitness scores. Initially, the algorithm trains an agent in a specific environment using a specific configuration, with the return in that particular task serving as a performance metric. Subsequently, the resulting policy, without retraining, is deployed in numerous other environments within the same class but with different configurations, which measures generalizability. These two independent metrics are computed multiple times for different random seeds, and stability is accounted for by penalizing algorithms that exhibit significant deviations across independent runs.

While this dissertation focuses on a specific use case, MetaPG can incorporate other real-world RL challenges as independent fitness scores. For instance:

- The area under the learning curve could quantify **sample efficiency**.
- Evaluating robustness against **high-dimensionality** could involve training a policy in a specific environment and then evaluating its performance in a scaled-up version of that environment. Alternatively, a second training run could be performed directly on the higher-dimensional environment, and the performance could be encoded as an independent fitness score.
- Incorporating evaluation environments where observations become non-stationary could capture the challenge of **non-stationarity**.
- A fitness score focusing on **delays** could be created by including evaluation environments with purposely slowed-down dynamics.
- Combining multiple real-world challenges into the same environment, akin to

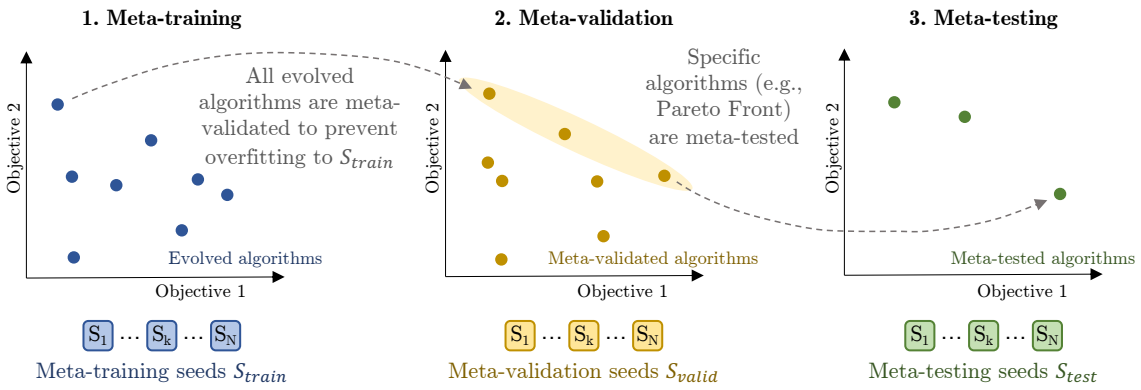



**Figure 5-5:** Running an experiment with MetaPG is divided into three phases. **1. Meta-training**: a population of algorithms is evolved using a set of random seeds $S_{train}$ to compute scores. **2. Meta-validation**: to prevent overfitting, the scores of all algorithms in the population are reevaluated using a different set of random seeds $S_{valid}$. **3. Meta-testing**: specific algorithms such as those in the Pareto Front are tested in different environments using a third set of random seeds $S_{test}$.

> [100], and registering the return could effectively identify new algorithms that are robust against **combined challenges**.

In MetaPG, fitness scores determine the Pareto Fronts that the method will discover, ultimately shaping the trade-offs the designer wishes to explore for the specific application at hand. Gaining a better understanding of these trade-offs assists the designer in selecting algorithms that align more closely with their preferences.

## 5.6 Running MetaPG experiments

Figure 5-5 describes the process of running experiments with MetaPG; these are divided into three phases: meta-training, meta-validation, and meta-testing. The rationale behind this division is to prevent overfitting to a specific set of random seeds during evolution. Specifically, each phase relies on a set of $N$ random seeds: $S_{train}$, $S_{valid}$, and $S_{test}$, respectively. Figure 5-5 considers an example with two objectives each encoded by a different fitness score. Each run of MetaPG begins with the meta-training phase, which consists of the evolution process described in the previous sections. The result of this phase is a population of evolved algorithms, each with a pair of meta-training fitness scores. Since the evolution process is non-deterministic, one could run each experiment multiple times without configuration changes and aggregate all resulting populations into one single larger population.

Then, to avoid selecting algorithms that overfit to the set of random seeds $S_{train}$, all algorithms in the population are reevaluated with a different set of seeds $S_{valid}$; this corresponds to the meta-validation phase. This phase provides updated fitness scores for all algorithms. In the absence of overfitting, there should not be much difference between meta-training and meta-validation scores, although the Pareto Front might differ from one phase to another. Finally, the meta-testing phase concerns assessing the fitness of specific algorithms we want to pick from the population (e.g., the algorithms in the Pareto Front) in different environments that may have or may have not been used during meta-training and meta-validation. To that end, a third set of seeds $S_{test}$ is used, which provides realistic fitness scores in the new environments.

## 5.7 Chapter summary and Contributions

This chapter has addressed two important research opportunities for real-world RL identified in the literature review: focusing on combined real-world RL challenges and the automation of the design process in RL. To that end, this chapter has presented MetaPG, an AutoML method that evolves new actor-critic loss functions represented as computational graphs that optimize multiple objectives. The design of these new loss functions is automated by means of evolutionary search and then, by considering multiple objectives separately, the method can encode metrics of robustness as fitness scores to guide the search.

Firstly, the chapter has delved into the related literature, highlighting that design automation is an emerging area of research that encompasses not only RL algorithms, but also their hyperparameters, policy networks, and certain elements of the environment. While evolutionary search for loss functions has been proposed before to optimize returns, no previous work has explored this approach in the context of optimizing for multiple RL objectives. Next, I have presented the main components of MetaPG and its overall workflow, which relies on the well-known multi-objective optimization algorithm NSGA-II.

The next sections of this chapter have each focused on different parts of MetaPG. First, I have discussed the search space used by MetaPG, which is based on typed nodes that encode input elements, a diverse set of operations, and the output elements corresponding to the losses. Although MetaPG's search space is high-dimensional, it offers significant flexibility to represent various classes of actor-critic algorithms. Next, the different elements of the evolution process have been discussed, including node and edge mutation, operation consistency checks, hashing to avoid redundant evaluations,

fitness scores that guide the search, population ranking, Pareto dominance, and hurdle evaluations to avoid investing excessive resources in poor algorithms.

Then, I have discussed the design of fitness scores for MetaPG. By considering complete training runs, numerous metrics that capture different aspects of robustness can be developed; I have provided different examples of such metrics. The last section has focused on running experiments with MetaPG, which are divided into three phases: meta-training, meta-validation, and meta-testing. Each phase utilizes a separate set of random seeds to prevent overfitting during evolution.

The contributions of this chapter are the following:

**Contribution 5.1** Proposed MetaPG, a method that combines multi-objective evolution and a search language representing actor-critic RL algorithms as graphs to discover new loss functions that optimize a set of different RL objectives.

**Contribution 5.2** Developed MetaPG's search space, which offers the flexibility to represent a wide range of actor-critic RL algorithms.

# Chapter 6

# Application of MetaPG to optimize performance, generalizability, and stability

This chapter applies MetaPG to a concrete use case of discovering new actor-critic RL algorithms that optimize for single-task performance, zero-shot generalizability, and stability across independent runs. Section **6.1** presents this use case, discusses its significance, and outlines the related work on optimizing for these goals. The definition of fitness scores for these objectives is addressed in Section **6.2**. Then, Section **6.3** focuses on the setups utilized in the different experiments. The experiments in this chapter are designed around two sets of environments. The first set consists of environments from the RWRL Environment Suite and OpenAI Gym and the experiments are presented in Section **6.4**. The second set of environments is based on the Brax physics simulators and its results are presented in Section **6.5**. Next, Section **6.6** dives deeper into the analysis of some of the evolved algorithms. Finally, Section **6.7** discusses the significance of the results and highlights specific areas of future work for MetaPG, and Section **6.8** summarizes the chapter and its main contributions.

## 6.1 Introduction

Chapter 5 introduced MetaPG, an evolutionary framework to discover new RL algorithms that optimize multiple RL objectives at the same time by means of independent fitness scores and a symbolic search language. This chapter puts MetaPG into practice by considering a use case in which both performance and generalizability

are optimized. Generalizability or generalization is one of the real-world challenges introduced in Chapter 2, it is present in many real-world domains and has been thoroughly studied in robotics. In addition, I also include the optimization of stability in this use case as an intrinsic metric; it is another real-world challenge identified in the literature review.

This chapter aims to answer the following questions:

1. Is MetaPG capable of evolving algorithms that improve upon performance, generalizability, and stability in different practical settings?
2. How well do discovered algorithms do in environments different from those used to evolve them?
3. Are the evolutionary results interpretable?

Generalizability and stability are two key aspects of RL robustness in real-world applications. One one hand, many real-world environments present themselves in multiple configurations (e.g., different sizes, structure, context, properties) and practitioners expect zero-shot generalization when facing these new configurations. Examples of this issue are present in robot manipulation [185], navigation [61], energy systems [304], and fluid dynamics [138]. On the other hand, real-world elements such as stochastic dynamics should not result in unstable learning behaviors that lead to undersired performance drops. Even for state-of-the-art RL algorithms, zero-shot generalization and instability are considerable challenges [100, 163].

Generalizability and stability are two suitable real-world challenges to be considered as a use case for MetaPG. In the literature, improving generalizability has been addressed by learning or encoding inductive biases in RL algorithms [310, 370]. Gains mostly come by manually modifying existing algorithms [69, 70, 186]. As environments become more complex and inductive biases become environment-specific, the cost of human-driven design might be too expensive when optimizing for generalizability [410], let alone when optimizing for both performance and generalizability [168], which are just two objectives we can consider but there are many more. In addition, stable algorithms that show consistent performance and generalizability across independent runs leads to higher algorithm reusability within the same environment and across environments.

To evaluate MetaPG for optimizing performance, generalizability, and stability, this section presents experiments using the zero-shot generalizability benchmark from the Real-World RL environment suite [100] and two environments from the Brax

physics simulator [126]. In this latter case, generalizability is accounted for by simulating perturbations like mass changes, different friction coefficients, and different joint torques. There is no specific benchmark for stability, the similarity between different learning curves is taken into account by using multiple random seeds and computing multiple performance and generalizability scores. In all cases, the Soft Actor-Critic (SAC) [152] graph presented in Section 5.3.3 is used to warm-start the population.

### 6.1.1 Related Work

Generalizability is one aspect of RL robustness that is not new in RL literature [210, 393]. Increasing generalizability has been addressed by means of environment randomization [14, 302, 360]. Other authors have shown that removing or adding certain algorithmic components impacts generalizability (e.g. using batch normalization [69], adding elements to rewards [56], or using regularizers [69, 186]). Other works directly achieve generalizability gains by modifying existing actor-critic RL algorithms [70, 310]. Vlastelica et al. [370] propose a hybrid architecture combining a neural network with a shortest path solver to find better inductive biases that improve generalizing to unseen environment configurations.

Achieving stable behaviors during training is essential in many domains, particularly in control applications [21, 185]. It has been shown that randomness can play a substantial role in the outcome of a training run [163]. Stable learning has been sought by means of algorithmic innovation [23, 125, 152, 193]. New stable algorithms have been mainly developed after looking into stability in isolation. For both stability and generalizability, new algorithms have been developed after looking at each challenge in isolation. MetaPG considers them jointly in addition to performance, ensuring that algorithm improvement directions that benefit the three metrics are explored.

## 6.2 Fitness scores

This chapter focuses on optimizing single-task performance, zero-shot generalizability, and stability across independent runs, and this section covers how each objective is encoded in the fitness scores. This applies to any of the phases described in Section 5.6, the only difference is the set of random seeds considered. Given an algorithm or graph to evaluate, the process to compute these fitness scores is depicted in Figure

6-1 and explained throughout this section. It relies on $N$ random seeds and a set of environments $\mathcal{E}$, which comprises multiple instances of the same environment class, including a training instance $E_{train} \in \mathcal{E}$. For example, $\mathcal{E}$ is the set of all *Cartpole* environments with different pole lengths (in the experiments these go from 0.1 meters to 3 meters in 10-centimeter intervals), and $E_{train}$ corresponds to an instance with a specific pole length (1 meter in the experiments).

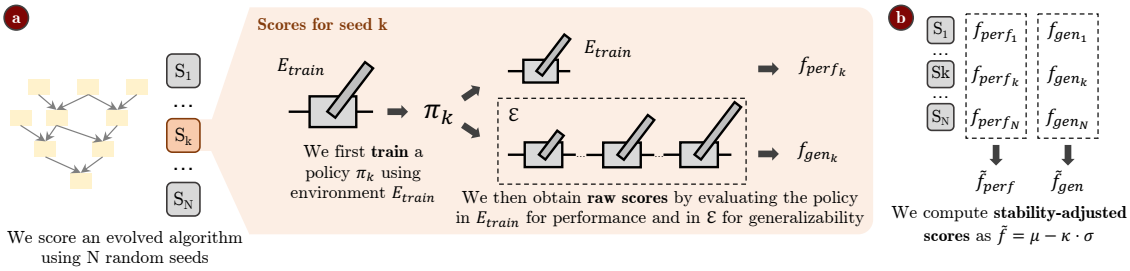



**Figure 6-1:** Process to compute fitness scores given an algorithm to evaluate. **(a)** Independently for each seed from a set of $N$ seeds, a policy $\pi_k$ is trained using the algorithm to evaluate, the environment $E_{train}$, and a seed $k$. During training, the policy is allowed to take stochastic actions. Then, $\pi_k$ is evaluated deterministically on that same environment $E_{train}$ to get a raw performance score $f_{perf_k}$, and on a set of environments $\mathcal{E}$ (same environment with different configurations; e.g., different pole lengths as shown) to get a raw generalizability score $f_{gen_k}$. **(b)** Stability-adjusted fitness scores $\tilde{f}_{perf}$ and $\tilde{f}_{gen}$ are computed by aggregating raw scores from each seed using Equation 6.3.

### 6.2.1 Raw performance and generalizability scores

The first step (see Figure 6-1a) is to compute raw performance and generalizability scores for each individual seed. Using $E_{train}$ and seed $k$ to train a policy $\pi_k$, the performance score $f_{perf_k}$ is the average evaluation return on the training environment configuration:

$$f_{perf_k} = \frac{1}{N_{eval}} \sum_{n=1}^{N_{eval}} G_n(\pi_k, E_{train}), \tag{6.1}$$

where $G_n$ corresponds to the normalized return (e.g., keeping returns between 0 and 1) for episode $n$ given a policy and an environment instance, and $N_{eval}$ is the number of evaluation episodes. Algorithms that learn faster in the training environment and overfit to it obtain higher performance scores.

The generalizability score $f_{gen_k}$ is in turn computed as the average evaluation return of the policy trained on $E_{train}$ over the whole range of environment configu-

rations. Generalizability addresses the real-world challenge of taking good actions in unseen instances of the environment, so the policy is trained on a single environment configuration (for example 1.0 meter pole length) and then is evaluated in a zero-shot fashion to new unseen environment configurations:[1]

$$f_{gen_k} = \frac{1}{|\mathcal{E}|N_{eval}} \sum_{E \in \mathcal{E}} \sum_{n=1}^{N_{eval}} G_n(\pi_k, E) \tag{6.2}$$

### 6.2.2 Stability-adjusted scores

The third objective in this use case is stability. While a third raw fitness score could be considered, in practice one looks for stable training results for both performance and generalizability. Therefore, I encode it intrinsically in the two other metrics and, from the optimization perspective, keep the problem as 2-objective. To that end, once there are independent raw scores for each seed ($N$ different performance and generalizability scores), the *stability-adjusted* scores (see Figure 6-1b) are defined as

$$\tilde{f} = \mu(\{f_n\}_{n=1}^{N}) - \kappa \cdot \sigma(\{f_n\}_{n=1}^{N}) \tag{6.3}$$

where $f$ is a score (performance or generalizability), $f_n$ denotes the score for seed $n$; $\mu$ and $\sigma$ are the mean and standard deviation across the $N$ seeds, respectively; and $\kappa$ is a penalization coefficient. Then, the final fitness of a graph is the tuple ($\tilde{f}_{perf}$, $\tilde{f}_{gen}$).

## 6.3 Experimental setups

This section describes the different experiments and analyses that are carried out throughout the rest of the chapter alongside their experimental setups. First, Section 6.4 presents the results on running MetaPG on a set of environments from the RWRL Environment Suite [100] and OpenAI Gym [43]. For each case, it describes the Pareto Fronts, the behavior of the best algorithms, the stability of the population, and the cross-environment performance. Then, Section 6.5 follows a similar procedure for environments based on the Brax physics simulator [126], emphasizing the meta-testing and cross-environment results of the best algorithms in the population. Lastly,

[1] More precisely, it should be $E \in \mathcal{E} \setminus \{E_{train}\}$. In practice, this makes no significant difference in the metric because the number of test configurations is normally around 30.

Section 6.6 focuses on the interpretability of the algorithms and presents an analysis on the graph structure of two of the evolved graphs.

The experimental setups that support the analyses in this section are the following:

**Training environments** The following environments are used: *Cartpole* and *Walker* from the RWRL Environment Suite, *Pendulum* from OpenAI Gym, and *Ant* and *Humanoid* from the Brax physics simulator. As it matters to compute generalizability, different instances of these environments are defined by varying the pole length in Cartpole, the thigh length in Walker, the pendulum length and mass in Pendulum, and, to mimic a practical setting, the mass, friction coefficient, and torque in Ant and Humanoid. See Appendix A.2.1 for the specifics.

**Meta-training details** The population and maximum graph size consist of 100 individuals and 80 nodes in the Brax environments, respectively, and 1,000 individuals and 60 nodes in the rest of the environments. All are initialized using SAC as a warm-start (see Section 5.3.3), which consists of 33 nodes. Additional operation nodes are added to each individual until reaching the maximum amount of 60 nodes. For RL algorithm evaluation, 10 different random seeds $S_{train}$ are used and the number of evaluation episodes $N_{eval}$ is fixed to 20. In the case of Brax, since training takes longer, the number of seeds is reduced to 4 different but $N_{eval}$ is increased to 32. As discussed in Section 5.6, repeats are carried out; specifically, there are 10 independent repeats of MetaPG when evolving on RWRL Cartpole, RWRL Walker, and Gym Pendulum; 5 repeats when evolving on Brax Ant; and 3 repeats for Brax Humanoid. Meta-training uses 100 TPU 1x1 v2 chips for 4 days in the case of Brax environments ($\sim$200K and $\sim$50K evaluated graphs in Ant and Humanoid, respectively), and 1,000 CPUs for 10 days in the rest of environments ($\sim$100K evaluated graphs per experiment). In all cases fitness scores are normalized to the range [0, 1] and $\kappa = 1$ in (6.3). Additional details are in Appendix A.2.2.

**Meta-validation details** Meta-validation utilizes a set of 10 random seeds $S_{valid}$, disjoint with respect to $S_{train}$. In the case of Brax environments, 4 meta-validation seeds are used. The same applies during meta-testing. In each case, the number of seeds achieves a good balance between preventing overfitting and having affordable evaluation time. The value of $N_{eval}$ during meta-validation and meta-testing matches the one used in meta-training.

**Hyperparameter tuning** The same fixed hyperparameters are used during all meta-training. Algorithms are also meta-validated using the same hyperparameters. In the case of Brax environments, hyperparameter tuning occurs for all algorithms during meta-validation; additional details can be found in Appendix A.2.4. We also hyperparameter-tune all baselines we compare our evolved algorithms against.

**RL Training details** The architecture of the networks (both actor and critic) corresponds to two-layer fully-connected networks or MLPs with 256 units each. Additional training details are presented in Appendix A.2.2.

**Mutation** During mutation, there is a 50% chance an individual undergoes node mutation and a 50% chance it undergoes edge mutation. During node mutation, there is a 50% chance of replacing one node, a 25% chance of replacing 2 nodes, a 12.5% chance of replacing 4 nodes, and a 6.25% chance of replacing 8 and 16 nodes, respectively. During edge mutation, only one edge in the graph is replaced randomly.

**Hashing** In the hashing process a fixed set of synthetic inputs with a batch size of 16 is used to compute the hash value for each graph. A new algorithm is evaluated if and only if there is not a cache hit after computing the hash.

## 6.4 Evolution in RWRL and OpenAI Gym environments

In this section MetaPG is used on the two environments from the RWRL Environment Suite, Cartpole and Walker, and on Pendulum, from OpenAI Gym. In all cases the population is initialized with SAC and then undergoes meta-training and meta-validation. I compare the resulting algorithms with an SAC implementation from ACME [171]. When running ACME SAC in any environment, first there is a hyperparameter tuning phase and the two configurations that lead to the best stability-adjusted performance and the best stability-adjusted generalizability are picked (in the figures and tables these are identified as ACME SAC HPT Perf and ACME SAC HPT Gen, respectively).

The order of the analyses is the following: Section 6.4.1 presents the evolution results when using MetaPG on RWRL Cartpole. This includes reporting on the best graphs at the Pareto Front and visualizing the behavior of the best performer and

the best generalizer. Sections 6.4.2 and 6.4.3 do the same for RWRL Walker and OpenAI Gym Pendulum, respectively. Then, Section 6.4.4 extends the analysis on the stability achieved by the evolved algorithms and lastly, Section 6.4.5 addresses the performance and generalizability of the evolved algorithms when transferred to different environment than those they were evolved in.

### 6.4.1 RWRL Cartpole

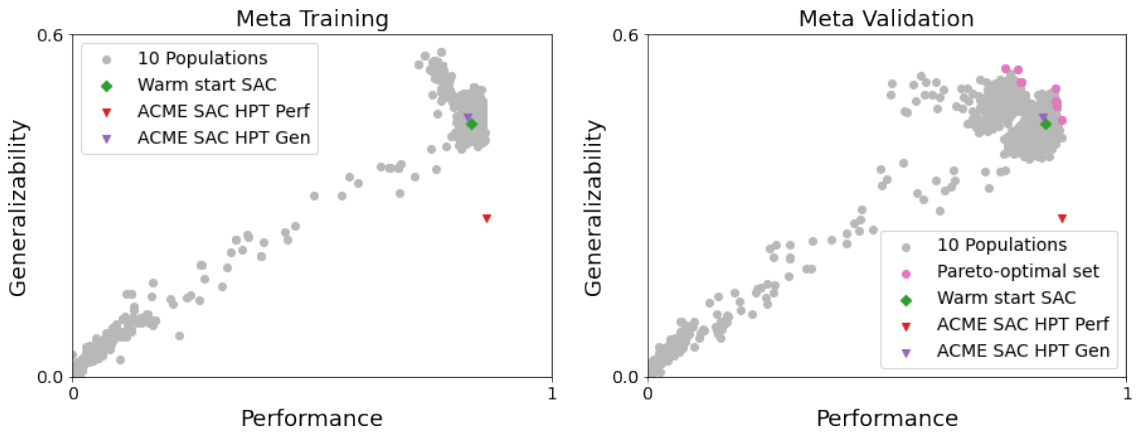



**Figure 6-2:** Meta-training and meta-validation stability-adjusted fitness scores (computed using Equation 6.3 across 10 seeds) for each RL algorithm in the population alongside the warm-start (SAC) and ACME SAC when using MetaPG on the RWRL Cartpole environment. The figure shows the meta-validated Pareto Front of algorithms that results after merging the 10 populations corresponding to the 10 repeats of the experiment.

Figure 6-2 depicts the meta-training and meta-validation populations after running MetaPG on RWRL Cartpole, with the meta-validated Pareto Front highlighted in pink. Then, Table 6.1 shows numeric values for each of the three metrics considered in this use case: performance, generalizability, and stability. In the case of stability, the table shows a measure of instability represented as the change in standard deviation compared to the warm-start (i.e., warm-start has the value 1.0). Instability is measured independently with respect to performance and generalizability.

The results from Table 6.1 show that MetaPG discovers RL algorithms that improve upon the warm-start's and ACME SAC's performance, generalizability, and stability in the same environment used during evolution. Compared to the warm-start, the best performer achieves a 4% improvement in the stability-adjusted performance score (from 0.836 to 0.868) and the best generalizer achieves a 20% increase

**Table 6.1:** Comparison of all algorithms in the Pareto Front with SAC (warm-start and hyperparameter-tuned ACME SAC) using metrics obtained in the RWRL Cartpole environment: average performance and generalizability, stability-adjusted performance and generalizability scores, and measure of instability (standard deviation $\sigma$ divided by the warm-start's $\sigma_{WS}$. These metrics are computed across 10 seeds and the best result in a *column* is **bolded**. Rows in gray correspond to algorithms that improve upon the warm-start in all metrics. †In contrast to performance and generalizability, the lower the instability the better.

| | **Performance** | | **Generalizability** | | **Instability**† ($\sigma/\sigma_{WS}$) | |
|---|---|---|---|---|---|---|
| **RL Algorithm** | $f_{perf}$ | $\tilde{f}_{perf}$ | $f_{gen}$ | $\tilde{f}_{gen}$ | Perf. | Gen. |
| **Pareto 1: Best performer** | **0.871** | **0.868** | 0.475 | 0.459 | 0.33 | **0.59** |
| **Pareto 2** | 0.857 | 0.856 | 0.513 | 0.488 | **0.11** | 0.93 |
| **Pareto 3** | 0.857 | 0.855 | 0.514 | 0.489 | 0.22 | 0.93 |
| **Pareto 4** | 0.856 | 0.854 | 0.517 | 0.493 | 0.22 | 0.89 |
| **Pareto 5** | 0.855 | 0.853 | 0.520 | 0.497 | 0.22 | 0.85 |
| **Pareto 6** | 0.854 | 0.852 | 0.531 | 0.514 | 0.22 | 0.63 |
| **Pareto 7** | 0.798 | 0.788 | 0.540 | 0.524 | 1.11 | **0.59** |
| **Pareto 8** | 0.794 | 0.784 | 0.546 | 0.528 | 1.11 | 0.67 |
| **Pareto 9** | 0.783 | 0.776 | 0.569 | 0.543 | 0.78 | 0.96 |
| **Pareto 10: Best generalizer** | 0.770 | 0.756 | **0.570** | **0.551** | 1.55 | 0.70 |
| **Warm-start SAC** | 0.845 | 0.836 | 0.487 | 0.460 | *1.0* | *1.0* |
| **ACME SAC HPT Perf** | 0.865 | 0.864 | 0.372 | 0.312 | **0.11** | 2.22 |
| **ACME SAC HPT Gen** | 0.845 | 0.833 | 0.518 | 0.478 | 1.33 | 1.48 |

in the stability-adjusted generalizability score (from 0.460 to 0.551). The table also highlights those algorithms in the Pareto Front that improve upon the warm-start in all metrics. For example, Pareto point 6 achieves a 2% and a 12% increase in both stability-adjusted performance and stability-adjusted generalizability, respectively. Then, in terms of the stability objective, the best performer reduces performance instability by 67% and the best generalizer achieves a reduction of 30% for generalizability instability.

The gains in generalizability and stability are substantial when comparing the results to hyperparameter-tuned ACME SAC. The best generalizer achieves a 15% increase in stability-adjusted generalizability compared to ACME SAC tuned for such metric. The instability in the hyperparameter-tuned SAC is twice as high (1.48 vs. 0.70, as shown in Table 6.1). In terms of performance, the best performer achieves a slightly better result compared to SAC hyperparameter-tuned for performance.

Figure 6-3 compares how the best performer and the best generalizer behave in different instances of the RWRL Cartpole environment in which the pole length

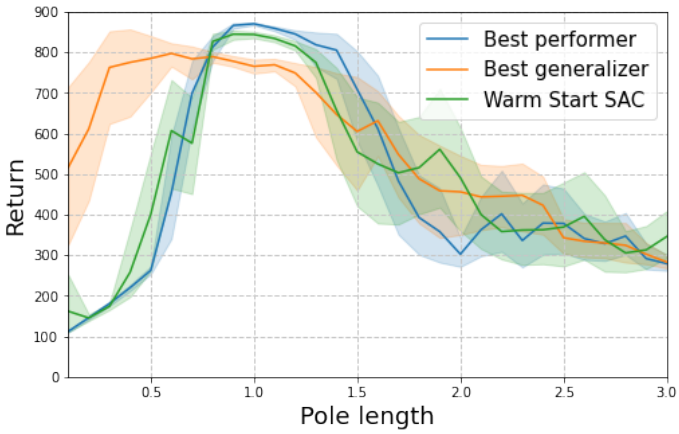



**Figure 6-3:** Average and standard deviation across seeds of the meta-validation performance of the best performer, the best generalizer, and the warm-start (SAC) when training on a single configuration of RWRL Cartpole and evaluating on multiple unseen ones. The pole length changes across environment configurations and a length of 1.0 is used as training configuration.

changes (all instances form the environment set $\mathcal{E}$ used during evolution). The goal of this analysis is to follow the same procedure described by [100]. The best performer achieves better return in the training configuration than the warm-start's. The best generalizer in turn achieves a lower return but it trades it for higher returns in configurations outside of the training regime, being better at zero-shot generalization.

Appendix C.1 updates Figure 6-3 by introducing PPO [328] in the comparison. One can observe that PPO is not well-suited for the continuous control tasks explored in this work. In contrast, SAC proves to be a good warm-start for the environments considered.

### 6.4.2 RWRL Walker

This section discusses the evolution results when running MetaPG with RWRL Walker as the training environment. Figures 6-4 and 6-5 show the resulting population and the performance across environment configurations for the best performer and the best generalizer in the Pareto Front, respectively. Exact numbers for each algorithm in the Pareto Front can be found in Table 6.2. Again, MetaPG finds a Pareto Front of evolved algorithms that outperform the warm-start in terms of both performance and generalizability. Note the warm-start is not hyperparameter-tuned at any point in the process, as a result, it might perform poorly, as is the case in this environment. Additional details on hyperparameter tuning in the RWRL environments can be found

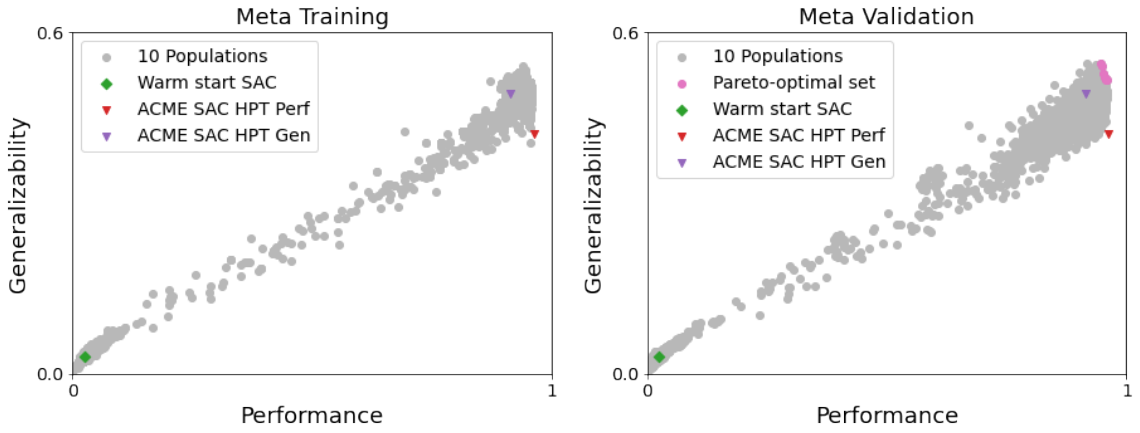



**Figure 6-4:** Meta-training and meta-validation stability-adjusted fitness scores (computed using Equation 6.3 across 10 seeds) for each RL algorithm in the population alongside the warm-start (SAC) and ACME SAC when using MetaPG on the RWRL Cartpole environment. The figure shows the meta-validated Pareto Front of algorithms that results after merging the 10 populations corresponding to the 10 repeats of the experiment.

**Table 6.2:** Comparison of all algorithms in the Pareto Front with SAC (warm-start and hyperparameter-tuned ACME SAC) using metrics obtained in the RWRL Walker environment: average performance and generalizability, stability-adjusted performance and generalizability scores, and measure of instability (standard deviation $\sigma$ divided by the warm-start's $\sigma_{WS}$. These metrics are computed across 10 seeds and the best result in a *column* is **bolded**. $^{\dagger}$In contrast to performance and generalizability, the lower the instability the better.

| | **Performance** | | **Generalizability** | | **Instability**$^{\dagger}$ $(\sigma/\sigma_{WS})$ | |
|---|---|---|---|---|---|---|
| **RL Algorithm** | $f_{perf}$ | $\tilde{f}_{perf}$ | $f_{gen}$ | $\tilde{f}_{gen}$ | Perf. | Gen. |
| **Pareto 1: Best performer** | 0.963 | 0.961 | 0.544 | 0.526 | 1.0 | 18.0 |
| **Pareto 2** | 0.962 | 0.959 | 0.536 | 0.524 | **1.0** | 12.0 |
| **Pareto 3** | 0.960 | 0.958 | 0.542 | 0.527 | 1.5 | 15.0 |
| **Pareto 4** | 0.959 | 0.956 | 0.541 | 0.528 | **1.0** | 13.0 |
| **Pareto 5** | 0.960 | 0.955 | 0.541 | 0.532 | 1.5 | 9.0 |
| **Pareto 6** | 0.954 | 0.951 | 0.555 | 0.546 | 2.5 | 9.0 |
| **Pareto 7: Best generalizer** | 0.955 | 0.950 | **0.569** | **0.554** | 2.5 | 15.0 |
| **Warm-start SAC** | 0.028 | 0.026 | 0.033 | 0.032 | ***1.0*** | ***1.0*** |
| **ACME SAC HPT Perf** | **0.968** | **0.965** | 0.444 | 0.430 | 1.5 | 14.0 |
| **ACME SAC HPT Gen** | 0.926 | 0.918 | 0.510 | 0.498 | 4.0 | 12.0 |

in Appendix A.2.3.

Since the ACME SAC versions are hyperparameter-tuned for both performance

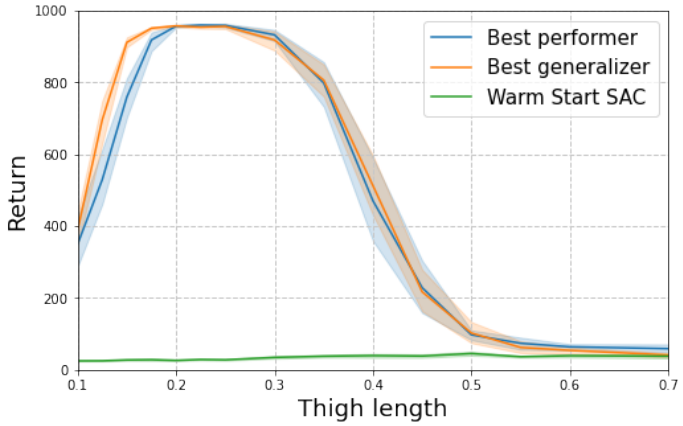



**Figure 6-5:** Average and standard deviation across seeds of the meta-validation performance of the best performer, the best generalizer, and the warm-start (SAC) when training on a single configuration of RWRL Walker and evaluating on multiple unseen ones. The thigh length changes across environment configurations and a length of 0.225 is used as training configuration.

and generalizability, respectively, their results improve substantially with respect to the warm-start SAC. While MetaPG does not offer improvements on performance in this case, it achieves a substantial generalizability increase (stability-adjusted), as the metric improves by 11% (from 0.498 to 0.554). This reinforces the ability of MetaPG to address an important real-world challenge that is encoded as a fitness score.

### 6.4.3 OpenAI Gym Pendulum

Third, this section presents the evolution results when running MetaPG with OpenAI Gym Pendulum as the training environment. Figures 6-6 and 6-7 show the resulting population and the performance across environment configurations for the best performer and the best generalizer in the Pareto Front, respectively. In the case of Pendulum, the generalizability fitness score is computed across the perturbation of two different parameters: the pendulum mass and the pendulum length. These parameters are changed separately, as opposed to varying both the mass and length of the pendulum in the same run. Exact numbers can be found in Table 6.3. The table compares the 8 algorithms in the Pareto Front with the warm-start SAC and hyperparameter-tuned versions of ACME SAC.

In the case of Pendulum, MetaPG offers substantial improvements in generalizability and stability, alongside a minor advantage in performance. The best generalizer achieves 15% more stability-adjusted generalizability with respect to the warm-start

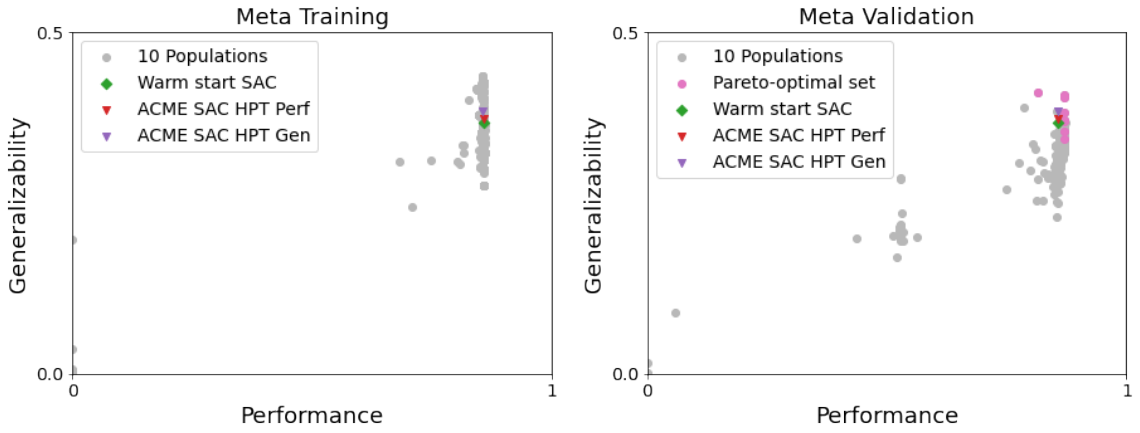



**Figure 6-6:** Meta-training and meta-validation stability-adjusted fitness scores (computed using Equation 6.3 across 10 seeds) for each RL algorithm in the population alongside the warm-start (SAC) and ACME SAC when using MetaPG on the OpenAI Gym Pendulum environment. The figure shows the meta-validated Pareto Front of algorithms that results after merging the 10 populations corresponding to the 10 repeats of the experiment.

**Table 6.3:** Comparison of all algorithms in the Pareto Front with SAC (warm-start and hyperparameter-tuned ACME SAC) using metrics obtained in the OpenAI Gym Pendulum environment: average performance and generalizability, stability-adjusted performance and generalizability scores, and measure of instability (standard deviation $\sigma$ divided by the warm-start's $\sigma_{WS}$. These metrics are computed across 10 seeds and the best result in a *column* is **bolded**. Rows in gray correspond to algorithms that improve upon the warm-start in all metrics. †In contrast to performance and generalizability, the lower the instability the better.

| | **Performance** | | **Generalizability** | | **Instability**† ($\sigma/\sigma_{WS}$) | |
|---|---|---|---|---|---|---|
| **RL Algorithm** | $f_{perf}$ | $\tilde{f}_{perf}$ | $f_{gen}$ | $\tilde{f}_{gen}$ | Perf. | Gen. |
| **Pareto 1: Best performer** | **0.887** | **0.877** | 0.360 | 0.349 | 0.45 | 0.73 |
| **Pareto 2** | 0.885 | 0.876 | 0.381 | 0.364 | **0.41** | 1.13 |
| **Pareto 3** | **0.887** | 0.876 | 0.391 | 0.377 | 0.45 | 0.93 |
| **Pareto 4** | **0.887** | 0.876 | 0.392 | 0.379 | 0.45 | 0.87 |
| **Pareto 5** | **0.887** | 0.876 | 0.393 | 0.386 | **0.41** | 0.47 |
| **Pareto 6** | 0.886 | 0.876 | 0.433 | 0.415 | 0.45 | 1.20 |
| **Pareto 7** | 0.886 | 0.875 | 0.437 | 0.418 | 0.45 | 1.27 |
| **Pareto 8: Best generalizer** | 0.868 | 0.834 | **0.445** | **0.424** | 1.55 | 1.40 |
| **Warm-start SAC** | 0.879 | 0.857 | 0.383 | 0.368 | *1.0* | *1.0* |
| **ACME SAC HPT Perf** | 0.880 | 0.866 | 0.392 | 0.380 | 0.64 | 0.80 |
| **ACME SAC HPT Gen** | 0.879 | 0.865 | 0.400 | 0.391 | 0.64 | **0.60** |

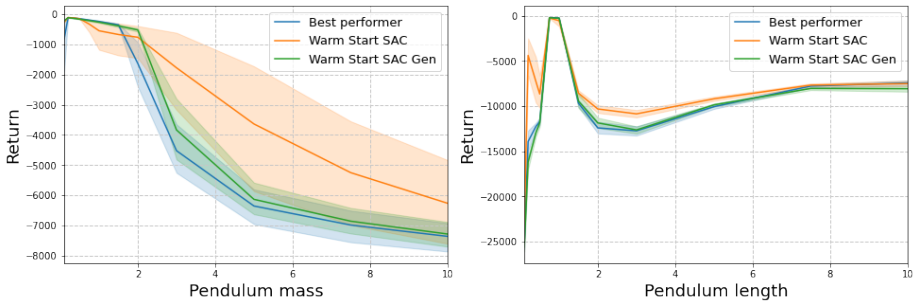



**Figure 6-7:** Average and standard deviation across seeds of the meta-validation performance of the best performer, the best generalizer, and the warm-start (SAC) when training on a single configuration of OpenAI Gym Pendulum and evaluating on multiple unseen ones. The pendulum mass and the pendulum length independently change across environment configurations (only one changes at a time). The training configurations use a pendulum mass and a pendulum length of 1.0 and 1.0, respectively.

(0.424 vs. 0.368) and a 8% improvement over ACME SAC (0.424 vs. 0.391). Figure 6-7 offers a clear view at the better robustness of the best generalizer with respect to the warm-start when varying the pendulum mass and its lenght. Looking at the stability itself, the majority of the algorithms show better stability, especially those with better performance. For example, the best performer achieves a 55% reduction in the instability of the performance metric and a 27% reduction in the instability of the generalizability metric. Finally, the advantage in stability-adjusted performance is 2% (0.877 vs. 0.857). This aligns with the observations from Chapter 2, which emphasized that RL design is currently good at performance instead of robustness, which offers more margin of improvement for the latter.

### 6.4.4 Stability analyses

This section presents additional figures focused on stability in order to complement the analyses carried out in the previous sections. As introduced in Section 6.2, stability is accounted for by penalizing the standard deviation across seeds, following Equation 6.3. Figures 6-8, 6-9, and 6-10 show how that impacts the distribution of individuals in the metrics space for RWRL Cartpole, RWRL Walker, and OpenAI Gym Pendulum. Specifically, for each environment, a subset of meta-validated graphs is selected and plotted in the metrics space as a data point symbolizing the average raw performance and raw generalizability achieved (i.e., no penalization for standard deviation) and an ellipse in which the length of each semiaxis corresponds to the standard deviation

for that metric. One can observe that algorithms in the Pareto Front and those closer to it present lower variability (i.e., smaller ellipses), emphasizing that MetaPG is successful in improving the stability of RL algorithms.

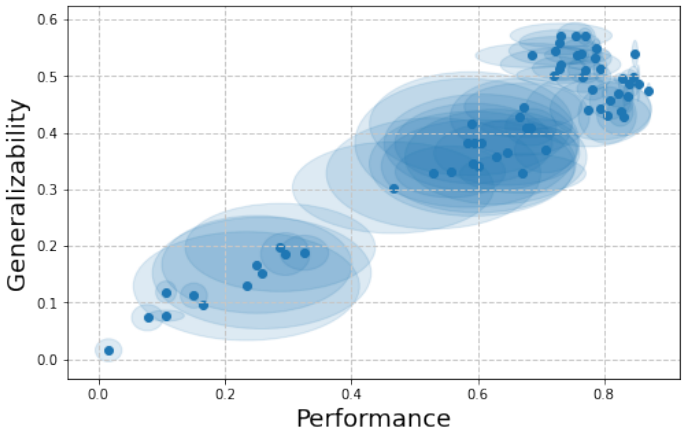



**Figure 6-8:** Meta-validation average fitness scores surrounded by an ellipse with semiaxes representing the standard deviation across seeds for each fitness score after running MetaPG on RWRL Cartpole. Only a subset of the graphs is shown to avoid cluttering.

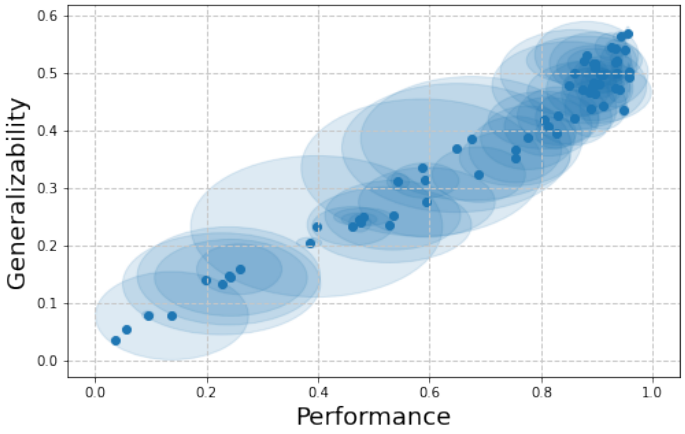



**Figure 6-9:** Meta-validation average fitness scores surrounded by an ellipse with semiaxes representing the standard deviation across seeds for each fitness score after running MetaPG on RWRL Walker. Only a subset of the graphs is shown to avoid cluttering.

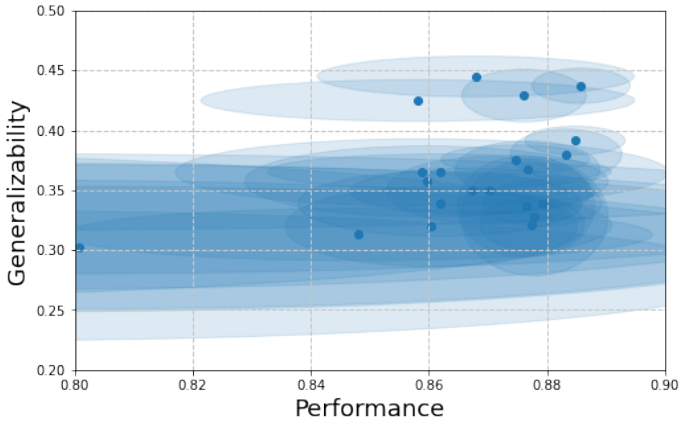



**Figure 6-10:** Meta-validation average fitness scores surrounded by an ellipse with semiaxes representing the standard deviation across seeds for each fitness score after running MetaPG on OpenAI Gym Pendulum. Only a subset of the graphs is shown to avoid cluttering.

### 6.4.5 Transferring evolved algorithms across environments

The last step of the analyses on the RWRL Environment Suite and OpenAI Gym is to evaluate how the algorithms evolved in one specific environment perform in the other environments. To that end, hyperparameter tuning is used during the transfer (see Appendix A.2.3). To better explain this analysis, I take the case of RWRL Cartpole, which is presented in Table 6.4. The table is divided into two sets of columns: best performance and best generalizability. The best performance columns correspond to the stability-adjusted performance and generalizability scores of algorithms that have been hyperparameter-tuned for stability-adjusted performance. Conversely, the best generalizability columns correspond to the stability-adjusted fitness scores of algorithms that have been hyperparameter-tuned for stability-adjusted generalizability.

Then, there are several rows. One of the rows correspond to the algorithms evolved directly on RWRL Cartpole, and it shows the scores for the best performer (on the best performance columns) and for the best generalizer (on the best generalizability columns), all scores copied from Table 6.1. Then, for both RWRL Walker and OpenAI Gym Pendulum, the table represents one row for each best performer and best generalizer (e.g., the *Walker Perf.* corresponds to the algorithm that obtained the best stability-adjusted performance when evolving on RWRL Walker). For each of these algorithms, hyperparameter tuning is carried out, once maximizing stability-adjusted performance, and a second time maximizing stability-adjusted generalizability. The

scores resulting from the tuning process are reported on the left-hand side columns in the former case and on the right-hand side for the latter case. Additionally, there is one final row corresponding to the best stability-adjusted performance and generalizability of ACME SAC (also taken from Table 6.1). Tables 6.5 and 6.6 do the same analysis for RWRL Walker and OpenAI Gym Pendulum, respectively.

**Table 6.4:** Transfer results on RWRL Cartpole. The row highlighted in gray corresponds to the results of the evolution experiment in RWRL Cartpole. The rest correspond to the best stability-adjusted performance and stability-adjusted generalizability configurations that result from doing hyperparameter tuning to the best performer and the best generalizer evolved in different environments. For example, the best performer evolved in OpenAI Gym Pendulum (row Pendulum Perf.) achieves 0.856 stability-adjusted performance and 0.478 stability-adjusted generalizability after being hyperparameter-tuned for the former metric. When that same algorithm is hyperparameter-tuned for generalizability, the results are shown on the right-hand side of the table.

| | **RWRL Cartpole** | | | |
|---|---|---|---|---|
| | **Best performance** | | **Best generalizability** | |
| **RL Algorithm** | $\tilde{f}_{perf}$ | $\tilde{f}_{gen}$ | $\tilde{f}_{perf}$ | $\tilde{f}_{gen}$ |
| **Cartpole** | 0.868 | 0.459 | 0.756 | 0.551 |
| **Walker Perf.** | 0.839 | 0.403 | 0.802 | 0.415 |
| **Walker Gen.** | 0.576 | 0.335 | 0.576 | 0.335 |
| **Pendulum Perf.** | 0.856 | 0.478 | 0.846 | 0.523 |
| **Pendulum Gen.** | 0.804 | 0.438 | 0.687 | 0.469 |
| **ACME SAC** | 0.864 | 0.312 | 0.833 | 0.478 |

The results vary slightly across environments, although there are several interesting trends. In general, the best performers transfer better to new environments; they obtain better scores than the best generalizers after doing hyperparameter-tuning, independently on whether the tuning is for performance or generalizability. A possible reason for that is that generalizability is environment-specific so, in order to achieve it, best generalizers learn many aspects of the evolution environments, which are hard to transfer unless the new environment shares the dynamics. Even though best performers transfer reasonably well, their fitnesses are not better than ACME's in the majority of the cases, although they are close. This highlights that the experimental setup considered in this chapter does not favor finding algorithms that generalize well across many environments but become good performers and generalizers for a specific environment instead. In many applications, one might be interested to find

**Table 6.5:** Transfer results on RWRL Walker. The row highlighted in gray corresponds to the results of the evolution experiment in RWRL Walker. The rest correspond to the best stability-adjusted performance and stability-adjusted generalizability configurations that result from doing hyperparameter tuning to the best performer and best generalizer evolved in different environments. Read the caption of Table 6.4 for an example of how to read this table.

| | **RWRL Walker** | | | |
|---|---|---|---|---|
| | **Best performance** | | **Best generalizability** | |
| **RL Algorithm** | $\tilde{f}_{perf}$ | $\tilde{f}_{gen}$ | $\tilde{f}_{perf}$ | $\tilde{f}_{gen}$ |
| **Cartpole Perf.** | 0.955 | 0.477 | 0.952 | 0.521 |
| **Cartpole Gen.** | 0.029 | 0.031 | 0.024 | 0.034 |
| **Walker** | 0.961 | 0.526 | 0.950 | 0.554 |
| **Pendulum Perf.** | 0.466 | 0.230 | 0.466 | 0.230 |
| **Pendulum Gen.** | 0.905 | 0.465 | 0.903 | 0.483 |
| **ACME SAC** | 0.965 | 0.430 | 0.918 | 0.498 |

**Table 6.6:** Transfer results on OpenAI Gym Pendulum. The row highlighted in gray corresponds to the results of the evolution experiment in OpenAI Gym Pendulum. The rest correspond to the best stability-adjusted performance and stability-adjusted generalizability configurations that result from doing hyperparameter tuning to the best performer and best generalizer evolved in different environments. Read the caption of Table 6.4 for an example of how to read this table.

| | **Gym Pendulum** | | | |
|---|---|---|---|---|
| | **Best performance** | | **Best generalizability** | |
| **RL Algorithm** | $\tilde{f}_{perf}$ | $\tilde{f}_{gen}$ | $\tilde{f}_{perf}$ | $\tilde{f}_{gen}$ |
| **Cartpole Perf.** | 0.862 | 0.329 | 0.857 | 0.349 |
| **Cartpole Gen.** | 0.821 | 0.323 | 0.821 | 0.323 |
| **Walker Perf.** | 0.860 | 0.324 | 0.813 | 0.375 |
| **Walker Gen.** | 0.849 | 0.319 | 0.679 | 0.388 |
| **Pendulum** | 0.877 | 0.349 | 0.834 | 0.424 |
| **ACME SAC** | 0.865 | 0.380 | 0.865 | 0.391 |

specialized components that can offer good results for the application considered. The next section extends the analyses on transferrability using the Brax environments as support.

## 6.5 Evolution in Brax environments

In this second set of experiments, the goal is to test MetaPG on more intricate environments in which dynamics are more complex but similar among environments. These environments are *Ant* and *Humanoid* from the Brax physics simulator [126], which allows running environment transitions on GPUs and TPUs using JAX [42]. For this environment, generalizability is assessed by adding three different perturbations to the environments in the form of changes in the friction coefficient, the body mass, and the torque (see Appendix A.2.1 for detailed descriptions).

Given the complexity of the environments and the added runtime, in these experiments I meta-train the population, then meta-validate the best few individuals, and then meta-test them before comparing. The warm-start SAC is also hyperparameter-tuned (independently of evolution) and used in the comparison. Transferring across environments is also evaluated in this case, using the hyperparameter tuning process described in Appendix A.2.4. Section 6.5.1 presents the results for Brax Ant and Section 6.5.2 does the same for Brax Humanoid.

### 6.5.1 Brax Ant

Figure 6-11 shows the behavior of evolved algorithms when meta-tested in perturbed Brax Ant environments. Algorithms are first evolved independently in both Ant and Humanoid, and then those algorithms that have the highest stability-adjusted performance score $\tilde{f}_{perf}$ during meta-training are selected. Then, using the meta-validation seeds, the algorithms are hyperparameter-tuned for generalizability (i.e., Figure 6-11 shows the meta-testing curves for the algorithms that were hyperparameter-tuned for generalizability), and then re-evaluated using the meta-testing seeds. In all cases, the algorithms are also compared against the warm-start SAC. Table 6.7 shows the full meta-testing fitness scores.

These results highlight that an algorithm evolved by MetaPG in Brax Ant performs and generalizes better than a SAC baseline in the same environment. Specifically, an improvement of 15% in stability-adjusted performance (0.729 vs. 0.632) and of 10% in stability-adjusted generalizability (0.592 vs. 0.537) are observed. In addition, there is a 23% reduction in instability. Furthermore, one can observe that an algorithm initially evolved using Brax Humanoid and meta-validated in Brax Ant transfers reasonably well to Brax Ant during meta-testing, achieving a slight loss of performance compared to hyperparameter-tuned SAC. Fewer graphs were evolved in

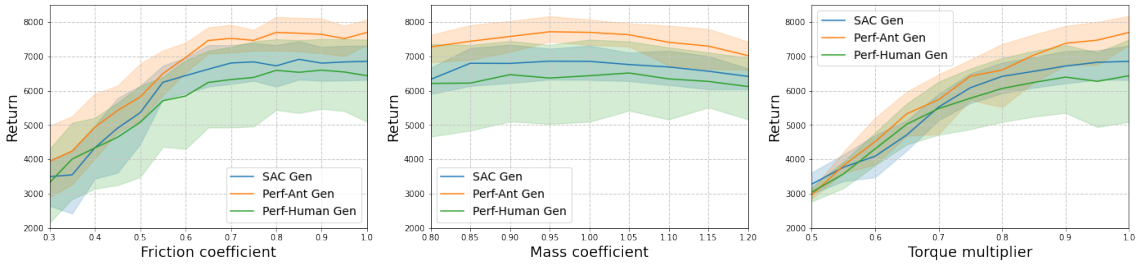



**Figure 6-11:** Comparison of the average and standard deviation across of the meta-testing evaluation of three algorithms after hyperparameter tuning: a loss function evolved in Brax Ant (orange), a loss function evolved in Brax Humanoid (green, to assess environment transfer), and the SAC baseline used as warm-start (blue). Each figure corresponds to a parameter of the environment being altered including the friction coefficient, the mass coefficient, and the torque multipliers. In all cases 1.0 is used as training configuration and to evaluate performance and the rest are used to assess generalizability.

**Table 6.7:** Meta-tested performance and generalizability scores (average and stability-adjusted), and measure of instability (standard deviation $\sigma$ divided by the warm-start's $\sigma_{WS}$) for algorithms first evolved in Brax Ant and Brax Humanoid, and then evaluated on Brax Ant on a different set of seeds. Scores compared against the hyperparameter-tuned warm start SAC. Meta-testing uses 4 seeds and the best result in a *column* is **bolded**.

| | **Performance** | | **Generalizability** | | **Instability**$^{\dagger}$ $(\sigma/\sigma_{WS})$ | |
|---|---|---|---|---|---|---|
| **RL Algorithm** | $f_{perf}$ | $\tilde{f}_{perf}$ | $f_{gen}$ | $\tilde{f}_{gen}$ | Perf. | Gen. |
| **Ant performer** | **0.770** | **0.729** | **0.627** | **0.592** | **0.77** | **0.97** |
| **Humanoid performer** | 0.643 | 0.526 | 0.553 | 0.467 | 2.21 | 2.39 |
| **Warm-start SAC** | 0.685 | 0.632 | 0.573 | 0.537 | *1.0* | *1.0* |

the case of Humanoid (50k compared to 200k evolved graphs for Ant), as training a policy in Brax Humanoid is more costly. Therefore, it is expected these results could improve if more algorithms were evolved in the population.

### 6.5.2 Evolution results for Brax Humanoid

Similar to last section, Figure 6-12 shows the behavior of evolved algorithms when meta-tested on Brax Humanoid, using the same environment perturbations described in Appendix A.2.1. Again, algorithms are first independently evolved in both Humanoid and Ant and, then, the algorithms with best stability-adjusted performance

$\tilde{f}_{perf}$ during meta-training are chosen. The next step is to meta-validate each algorithm considering multiple hyperparameter sets (see Appendix A.2.4) and pick those configurations that lead to the best stability-adjusted generalizability. Finally, all algorithms are meta-tested with fixed hyperparameters. Figure 6-12 shows performance across perturbations for the hyperparameter set that led to the best stability-adjusted generalizability and Table 6.8 reports on all meta-testing fitness scores, including the warm-start SAC (the warm-start is also hyperparameter-tuned independently).

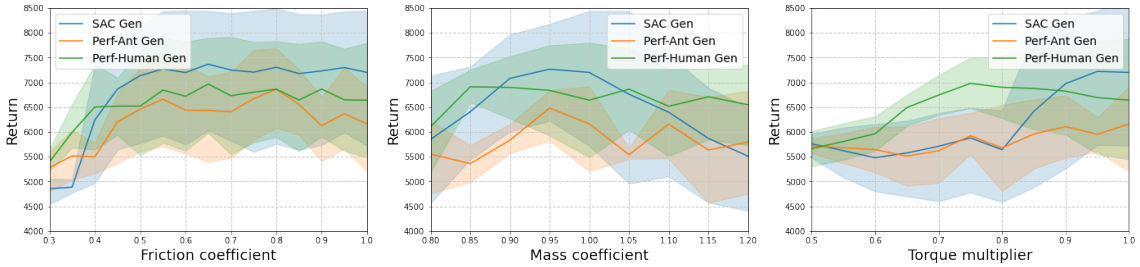



**Figure 6-12:** Comparison of the average and standard deviation across of the meta-testing evaluation of three algorithms after hyperparameter tuning: a loss function evolved in Brax Humanoid (green), a loss function evolved in Brax Ant (orange, to assess environment transfer), and the SAC baseline used as warm-start (blue). Each figure corresponds to a parameter of the environment being altered including the friction coefficient, the mass coefficient, and the torque multipliers. In all cases 1.0 is used as training configuration and to evaluate performance and the rest are used to assess generalizability.

**Table 6.8:** Meta-tested average and stability-adjusted performance and generalizability scores, and measure of instability (standard deviation $\sigma$ divided by the warm-start's $\sigma_{WS}$) for algorithms first evolved in Brax Humanoid and Brax Ant, and then evaluated on Brax Humanoid on a different set of seeds. We compare these scores against the hyperparameter-tuned warm start SAC. We compute these metrics across 4 seeds and the best result in a *column* is **bolded**.

| | **Performance** | | **Generalizability** | | **Instability**$^{\dagger}$ $(\sigma/\sigma_{WS})$ | |
|---|---|---|---|---|---|---|
| **RL Algorithm** | $f_{perf}$ | $\tilde{f}_{perf}$ | $f_{gen}$ | $\tilde{f}_{gen}$ | Perf. | Gen. |
| **Ant performer** | 0.440 | 0.374 | 0.420 | 0.384 | **0.63** | **0.51** |
| **Humanoid performer** | 0.474 | 0.391 | **0.462** | **0.420** | 0.80 | 0.60 |
| **Warm-start SAC** | **0.514** | **0.410** | 0.450 | 0.380 | *1.0* | *1.0* |

The numerical results of this analysis show that, while the algorithm evolved in Brax Humanoid achieves better stability-adjusted generalizability and a clear reduc-

tion in instability (up to 40%) compared to the warm-start SAC, it performs worse than SAC in meta-testing. This is a consequence of hyperparameter-tuning the algorithms for generalizability, hence the higher score. In addition, note fewer graphs are evolved in the specific case of Brax Humanoid (50K compared to 200K evolved graphs for Brax Ant), as training a policy in Humanoid is more costly. These results could improve if more algorithms were evolved in the population. Then, similar to the case meta-testing was carried out on Brax And, good cross-environment fitness is achieved compared to SAC, although the scores are lower. However, in this case the algorithm evolved in Brax Ant shows more stability than SAC when both are evaluated in Humanoid.

## 6.6 Analyzing evolved algorithms

This sections analyzes some of the evolved algorithms to gain insights on the evolution process. Specifically, it focuses on the experiments on RWRL Cartpole presented in Section 6.4.1, the equations for the experiments using the rest of the environments can be found in Appendix C.2. The experiment began by taking the warm-start SAC presented in Section 5.3.3, which has a policy loss and a critic loss. Now let's focus on the best performer and best generalizer from the meta-validation phase (first and last algorithms in the Pareto Front from Table 6.1). The policy loss $L_\pi$ and critic losses $L_{Q_i}$ (one for each critic network $Q_i$) observed from the graph structure for the best performer are the following:

$$L_\pi^{perf} = \mathbb{E}_{(s_t,a_t,s_{t+1})\sim\mathcal{D}}\left[\log(\min(\pi(\tilde{a}_{t+1}|s_{t+1}),\gamma)) - \min_i Q_i(s_t,\tilde{a}_t)\right] \tag{6.4}$$

$$L_{Q_i}^{perf} = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left(r_t + \gamma\left(\min_i Q_{targ_i}(s_{t+1},\tilde{a}_{t+1}) - \log\pi(\tilde{a}_{t+1}|s_{t+1})\right) - Q_i(s_t,a_t)\right)^2\right] \tag{6.5}$$

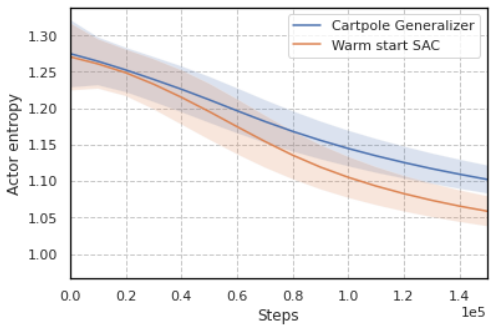
where $\tilde{a}_t \sim \pi(\cdot|s_t)$, $\tilde{a}_{t+1} \sim \pi(\cdot|s_{t+1})$, and $\mathcal{D}$ is an experience dataset extracted from the replay buffer. The changes and additions with respect to SAC are highlighted in blue, and in red the elements of the SAC loss function that evolution removes. Then, the loss equations for the best generalizer are:

$$L_\pi^{gen} = \mathbb{E}_{(s_t,a_t,s_{t+1})\sim\mathcal{D}}\left[\log\pi(\tilde{a}_t|s_t) - \min_i Q_i(s_{t+1},\tilde{a}_t)\right] \tag{6.6}$$

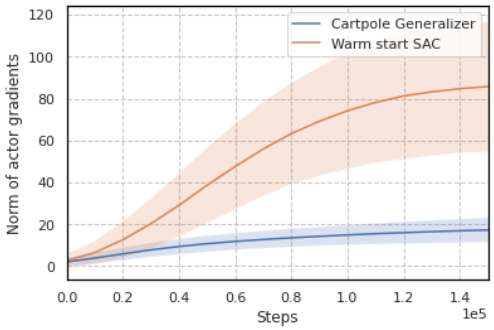
$$L_{Q_i}^{gen} = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\text{atan}\left(\left(r_t + \gamma\left(\min_i Q_{targ_i}(s_{t+1},\tilde{a}_t) - \log\pi(\tilde{a}_t|s_t)\right) - Q_i(s_t,a_t)\right)^2\right)\right] \tag{6.7}$$

While both algorithms resemble the warm-start SAC, one can observe that the best performer does not include the entropy term in the critic loss while the best generalizer does (i.e., they correspond to setting $\alpha$ to 0 and 1 in the original SAC

algorithm [152], respectively). This aligns with the hypothesis that, since ignoring the entropy pushes the agent to exploit more and explore less, the policy of the best performer overfits better to the training configuration compared to SAC. In contrast, the best generalizer is able to explore more. Figure 6-13a validates the latter observation showing a higher entropy for the best generalizer's actor compared to the warm-start's.




**(a)** Average entropy of the policy during training for RWRL Cartpole.

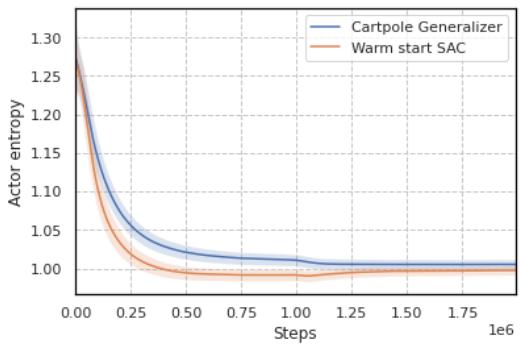



**(b)** Average gradient norm of the actor loss during training for RWRL Cartpole.

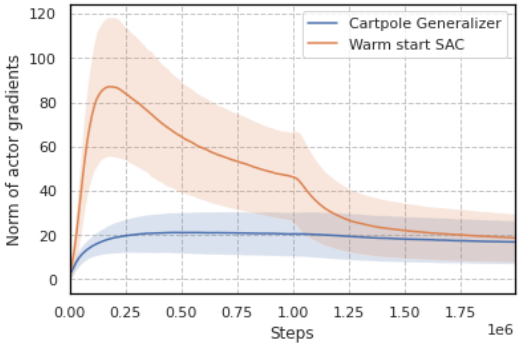
**Figure 6-13:** Analysis of the entropy and gradient norm of the actor when evaluating the best generalizer from RWRL Cartpole in comparison to the warm-start. An hypothesis is that this increase in entropy and decrease in gradient norm with respect to SAC contribute to achieve better generalizability.

The use of arctangent in the critic loss of the best generalizer is also noticeable as, supported by Figure 6-13b, this operation serves as a way of clipping the loss, which makes gradients smaller and thus prevents the policy's parameters from changing too abruptly. In the experiments of this dissertation, the number of training episodes (when evaluating a specific algorithm) is fixed as a compromise between achievable returns and low evaluation runtimes. As a consequence, clipping the loss has an early-stopping effect compared to the baseline and results in a policy less overfitted, which benefits generalizability.

To better understand this effect, Figure 6-14 shows the same plots than Figure 6-13 but in this case the training budget for the agent is more than an order of magnitude larger. One can observe that ignoring the fixed number of training episodes and letting agents train for longer makes metrics like the entropy and the gradient norm converge to similar values with respect to the warm-start. While training until convergence is usually preferred, in certain applications the number of training episodes might be a constraint, so MetaPG's ability to exploit this kind of constraints is beneficial in those setups.




**(a)** Average entropy of the policy during training for RWRL Cartpole.




**(b)** Average gradient norm of the actor loss during training for RWRL Cartpole.

**Figure 6-14:** Analysis of the entropy and gradient norm of the actor when evaluating the best generalizer from RWRL Cartpole in comparison to the warm-start and increasing the number of training episodes 13x with respect to Figure 6-13.

## 6.7 Discussion

This chapter has shown that MetaPG can discover novel RL loss functions that achieve better single-task performance, zero-shot generalizability, and training stability compared to warm-start algorithms such as SAC. MetaPG is able to automate a process that in practice takes many iterations (see Figure 6-15a). This section discusses some interesting areas of future work for MetaPG.

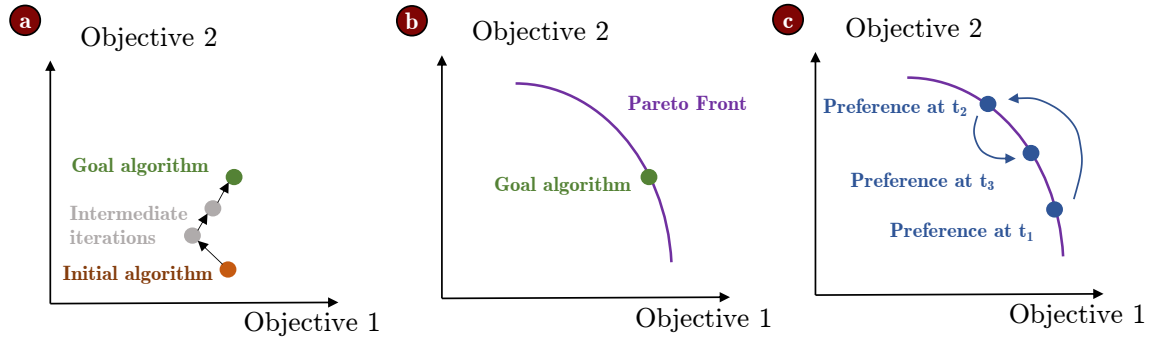



**Figure 6-15:** **(a)** In many practical contexts, achieving more than one objective is an empirical iteration-based process. **(b)** MetaPG's advantage is identifying trade-offs among objectives and, when the search converges, Pareto Fronts of algorithms. **(c)** During operation, preferences might change, resulting in some objectives traded for others.

**Additional use cases** One obvious area of future work for MetaPG is evaluating it on other use cases with different objectives and fitness scores. Given the limited

computing budget, this dissertation has focused on one specific use case, which already has proven useful to validate MetaPG and to gain significant understanding on the applicability of this method in the context of real-world RL. Other use cases could consider some of the example fitness scores described in Section 5.5.

**Search space complexity** The search space consits only of primitive operators, similar to the work in [67]. Although this promotes expressiveness of algorithms in the space, it requires many nodes and edges to represent a loss function which makes the search space vast, which entails that good loss functions are extremely sparse in this space. It is challenging for evolutionary algorithms to traverse this space given limited search budget. One area for improvement is to design a more efficient search space so that there is a greater chance of discovering better algorithms under the same computation budget.

Still, MetaPG's search's outcome is aligned with the community's practice to find better algorithms, as advances in RL often come from small variations in existing algorithms (e.g., CQL [223], S-DQN and M-DQN [367]), especially when addressing generalizability [69, 70, 186]. Hopefully, further work in the search process will lead to larger algorithmic changes. In earlier versions of MetaPG, with a reduced search space, it was able to evolve REINFORCE [348] from scratch.

One of MetaPG's advantages with respect to other methods is its capacity to identify Pareto Fronts among the considered objectives (see Figure 6-15b). However, current efforts to obtain Pareto Fronts require combining multiple evolution runs, since each individual run could converge to a different local optimum. Future work also includes improving evolution to avoid converging early to local optima.

**Cost efficiency** Running MetaPG involves a non-negligible upfront computational cost. However, this cost can be amortized by reusing the evolved algorithms; this is something that is aligned with the economy of scale AutoML approaches look for [300]. First, the cost is amortized by generating a Pareto Front of stable loss functions from which practitioners can choose a specific point based on the preference among objectives. A single-objective approach would require running MetaPG every time this preference changes (see Figure 6-15c). In addition, achieving cross-environment generalization provides an additional perspective on amortizing the cost; algorithms can be reused across different domains and environments. Finally, the results also show that the largest fitness jumps are attained in the earliest iterations, so one could run MetaPG for shorter time in some cases (see Appendix C.3).

**Transferrability of algorithms** Another direction of future work is to improve the transferrability of evolved algorithms to new domains. On one hand, the results have demonstrated that evolved algorithms transfer reasonably well (especially best performers in the original environment). On the other hand, sometimes they do not perform better than SAC in the new environments without hyperparameter tuning first. While hyperparameter tuning has not been used during evolution (which could suggest there is additional margin of improvement in terms of achieved scores), future work should address how fitness scores can be better aligned with cross-domain transferrability. At the same time, it poses an interesting research question of determining whether MetaPG is better suited to find "super algorithms" for specific environments or a new generation of all-purpose algorithms.

**Ensembling** It would also be interesting to ensemble the evolved loss functions on the Pareto front. Such loss function may give additional flexibility for practitioners when designing an RL system by encoding complex design choices into an interpolation across objectives.

**Exploiting training constraints** Finally, Section 6.6 has showed that the use of the arctangent in Equation (6.7) might benefit generalizability by serving as an early-stop before the policy overfits to the training configuration. The experiments fixed a certain number of training episodes as a compromise between achievable returns and evaluation runtimes. At the same time, letting run for longer makes certain metrics across algorithms converge to similar values. While training until convergence is usually preferred, in certain applications the number of training episodes might be a constraint, so MetaPG's ability to exploit this kind of constraints beneficial in those setups.

## 6.8 Chapter summary and Contributions

This chapter has focused on evaluating MetaPG on a specific use case on optimizing for single-task performance, zero-shot generalizability, and stability across independent training runs. The goal is not only to address the opportunities of design automation and combined challenges of real-world RL, but it also involves trade-off analysis as part of the design process in RL. This latter goal was another of the research opportunities identified in the second chapter of this dissertation.

The chapter has commenced by posing three specific questions to answer. I now repeat these questions with answers below each one:

1. **Is MetaPG capable of evolving algorithms that improve upon performance, generalizability, and stability in different practical settings?**

   Yes, MetaPG has found new RL algorithms that improve upon a popular baseline, SAC, in all objectives considered for different environments.

2. **How well do discovered algorithms do in environments different from those used to evolve them?**

   Evolved algorithms display different levels of cross-environment performance; algorithms that achieve the highest performance in the original environments have been found to transfer the best, achieving at least similar results compared to SAC, better in some cases.

3. **Are the evolutionary results interpretable?**

   By examining the equations of the evolved loss functions, interpretations of the performance and generalizability biases have been derived, proving that MetaPG provides valuable information about the search outcome that might not be accessible when using other AutoML methods such as black-box algorithms.

To answer these questions, this chapter has begun by presenting the specific fitness scores considered for this use case. These scores explicitly optimize performance and generalizability and implicitly optimize stability. Prior to that, related work on generalizability and stability has been reviewed. Then, I have presented the experimental setups considered, including information on training environments, meta-training and meta-validation configurations, and hyperparameter tuning protocols.

The first batch of experiments has focused on *Cartpole* and *Walker* from the RWRL Environment Suite and *Pendulum* from OpenAI Gym. The summary of the improvements achieved over the SAC warm-start due to evolution is reported in Table 6.9. In addition, these experiments have examined the transferrability across environments of different algorithms from Pareto Fronts, concluding that algorithms achieving high performance during evolution can transfer better.

Next, the results for the second batch of experiments have been outlined. These experiments have utilized the Brax physics simulator, specifically the *Ant* and *Humanoid* environments, and their evolution results are also presented in Table 6.9.

**Table 6.9:** Summary of improvements achieved by MetaPG over the warm-start SAC for each of the environments considered for this use case. † The performance of the warm-start was very poor in the case of RWRL Walker, comparisons are made with respect to ACME SAC instead.

| **Environment** | **Performance** | **Generalizability** | **Stability** |
|---|---|---|---|
| RWRL Cartpole | 4% | 20% | 67% |
| RWRL Walker† | 0% | 11% | 150% |
| OpenAI Gym Pendulum | 2% | 15% | 55% |
| Brax Ant | 15% | 10% | 23% |
| Brax Humanoid | 0% | 11% | 40% |
| **Average** | **4.2%** | **13.4%** | **67.0%** |

Additionally, this section has examined transferrability *with* hyperparameter tuning during meta-validation, which has demonstrated that algorithms achieve at least comparable performance with respect to SAC. The last section on experiments has focused on analyzing the structure of some of the evolved algorithms, showing that several of their substructures can be interpreted.

Finally, the chapter has concluded with a discussion on areas of future work for MetaPG. Given its complexity and the computational infrastructure needed, the limited experimental budget has left many interesting areas of future research unexplored, including improving the efficiency of the evolutionary search, making MetaPG more cost-efficient, and improving the transferrability of algorithms by means of new creative fitness scores.

The specific contributions of this chapter are the following:

**Contribution 6.1** Formulated MetaPG fitness scores to explicitly optimize for performance and generalizability and implicitly encourage stability.

**Contribution 6.2** By running MetaPG, identified set of Pareto-optimal loss functions that have been evolved in different environments and outperform Soft Actor-Critic for such environments in terms of performance, generalizability, and stability.

**Contribution 6.3** Provided, for the specific case of RWRL Cartpole, a comprehensive dataset[2] of algorithms obtained throughout the complete

[2]The dataset can be found at: https://github.mit.edu/garau/MetaPG-dataset

evolution process (not only the Pareto Front). This dataset may be further analyzed to gain additional insights on algorithmic changes and trade-offs.

# Chapter 7

# Designing DRL Systems for Specific Real-world Problems

This chapter supports shifting the focus from the emphasis on domain-agnostic aspects and methodologies for RL robustness addressed in the previous chapters to the upcoming exploration of concrete real-world applications in the remainder of the dissertation, which looks into aligning the design process of RL systems with the robustness priorities of these specific scenarios. To that end, Section **7.1** introduces the goals of this last part of the dissertation. Then, Section **7.2** discusses current domain-specific workflows when it comes to new real-world DRL methods and identifies gaps that are frequently overlooked. To better understand those gaps, this part of the dissertation relies on two use cases, which are introduced in Section **7.3**. Lastly, Section **7.4** concludes the chapter.

## 7.1 Introduction

As discussed in Chapter 2, a primary issue faced by domain-specific RL research is improving the development process of DRL systems so that robustness is prioritized. Currently, there exists an emphasis on performance that in many cases results in excessive tailoring to specific scenarios or subproblems. This approach, as argued in the implementation phase of the roadmap presented in Chapter 3, directly conflicts with system-level generalization and robustness, thereby creating impediments to effective deployment.

To investigate these issues more thoroughly, this dissertation's focus will now shift to analyzing specific real-world use cases. In Chapter 8, the robustness of DRL in

the context of frequency assignment for satellite communications will be examined. Subsequently, in Chapter 9, the study will extend to the problem of molecular optimization. Both use cases are outside of the realm of robotics, which aligns with another of the research opportunities this dissertation addresses.

This chapter effectively bridges the introductory aspects of how domain-specific RL research currently tackles the design and implementation of DRL systems for real-world applications, and this dissertation's subsequent focus on the two aforementioned use cases. To that end, this chapter initially introduces the workflow that domain-specific RL research typically follows when innovating on DRL for certain practical applications. The highlighted eight high-level steps help to frame the gaps that indicate a lack of robustness prioritization in the process. These gaps are the basis of the ensuing study in both use cases, which are also introduced in this chapter.

## 7.2 Understanding the workflow of domain-specific real-world RL research

I argue domain-specific RL research is guided by a multi-step workflow when developing new DRL systems for a particular real-world application. This workflow, while comprehensive, often overlooks several crucial steps necessary for prioritizing robustness in addition to performance. This section delves into the details of this workflow and presents such gaps.

The typical workflow that domain-specific RL research follows is depicted on the left side of Figure 7-1 and, at a high level, can be broken down into the following eight steps:

1. **Identify and scope the real-world problem to solve.** The first step simply involves picking a problem to solve. In many cases this will correspond to a problem already explored in the literature by means of other methods.
2. **Formulate the problem as a Markov Decision Process (MDP).** Unless the problem can be represented as a MDP involving sequential decision-making, it will be hard to design a DRL system for it. Therefore, the next step is to frame it as a MDP. For example, in the case of molecular optimization, one of the two use cases studied in this dissertation, the MDP might consist of generating the molecule atom by atom or scaffold by scaffold. There could be multiple ways of defining the MDP.

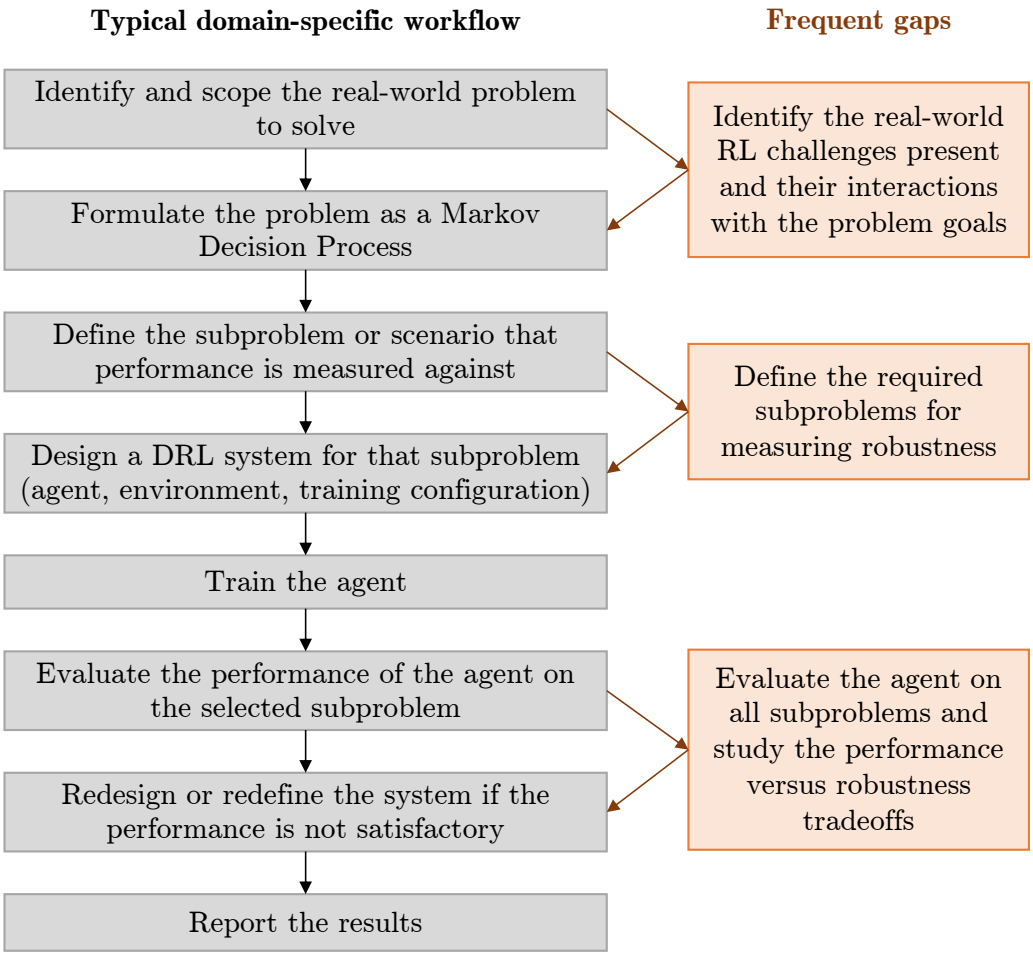



**Figure 7-1:** Workflow of domain-specific real-world DRL research, highlighting the common steps followed in the process (left) and the frequent gaps that need to be addressed for ensuring robustness (right).

3. **Define the subproblem or scenario that performance is measured against.** In Chapter 1 the concept of subproblems was introduced. While the MDP can be defined in a generic way, practitioners might have a specific subproblem in mind when designing the different components of the system. This might be motivated by a certain benchmark or environment they want to simulate. In the case of molecular optimization, this might correspond to optimizing for a specific property.

4. **Design a DRL system for that subproblem.** Next, the components of the system are designed, which includes selecting a state representation, action space, reward function, etc. At this point, the practitioner might consciously or subconsciously introduce inductive biases given by the subproblem considered. As a consequence, the design might be too tailored to that subproblem.

5. **Train the agent.** This step is typically straightforward, it requires training the agent and observing a learning behavior that indicates the system, based on its design, is successful at training the agent.
6. **Evaluate the performance of the agent on the selected subproblem.** There is always a procedure to evaluate and understand the performance of the agent. This can be based on standardized benchmarks or evaluation environments that are also designed. In any case, if there has been excessive tailoring to the subproblem under consideration, this performance measure might be too tied to that particular subproblem and therefore not sufficiently addressing robustness. In the case of molecular optimization, this might correspond to evaluating the agent at optimizing for that particular property, but not considering more realistic scenarios that entail a broad range of multi-property optimization tasks.
7. **Redesign or redefine the system if the performance is not satisfactory.** Many of the problems solved via DRL are hard in nature, therefore it is common to change some aspects of the design of the DRL system once the practitioner gets performance feedback. This step is generally only oriented at increasing the performance of the agent.
8. **Report the results.** Success stories generally get attention in the form of publications; there is value in proving DRL works for a certain subproblem. However, as discussed in Chapter 1, in most cases the lack of follow-through leaves the study of deployability unexplored for that particular problem.

This workflow describes a systematic approach to tackling real-world problems using DRL. However, it often misses several opportunities to take robustness into account in the development process. These can be summarized in three key steps that interleave some of the steps of the presented workflow, as shown on the right-hand side of Figure 7-1.

First, during the first and second steps, the domain-agnostic challenges of real-world RL are often overlooked. While only a few of the challenges might be present in the considered problem, their combined interaction is already detrimental enough for any DRL system that ignores them [100]. Therefore, not including these challenges as part of problem formulations and MDP descriptions can almost certainly lead to a mismatch between the performance during development and the performance at deployment. Capturing these challenges is as important as taking into account the domain specifics when formulating the MDP. For example, if partial observability will

be present at deployment, the MDP should be framed as a partially-observable MDP or POMDP.

Second, the third step of the workflow entails designing a subproblem or scenario that performance is measured on. This subproblem biases the posterior design of the overall DRL system. Since robustness is not considered, other subproblems are not accounted for in order to expand the scope of assessment of the system. For example, for one of the use cases studied in this dissertation, frequency assignment for satellite communications, all DRL works found in the literature evaluate agents in low-dimensional scenarios. This contrasts with the upcoming landscape of the industry, which is quickly moving towards high-dimensional constellations. An agent that cannot complete the task in those scenarios will not be ready for deployment; since robustness against high-dimensionality is not evaluated, it is not clear whether published models are well-suited.

Third, one can assume there will not be a single design that achieves the best performance and the best robustness for the problem considered; trade-offs are likely. However, unless robustness is accounted for and measured, it is not possible to study how these trade-offs impact the problem at hand. As discussed in Chapter 3, understanding the performance versus robustness trade-offs is crucial to get a comprehensive picture of what needs to be improved about the DRL system and which approach is more likely to yield such improvement. Therefore, assessing the trade-offs should be an integral part of the evaluation process and any redesign of the system should also include robustness considerations if necessary. In both use cases studied in Chapters 8 and 9, I discuss the trade-offs that are found in each case.

Addressing the three aforementioned gaps is crucial for the development of robust DRL systems. To ground these recommendations in real-world examples, in the next section I present the two real-world use cases this dissertation considers, frequency assignment for satellite constellations and molecular optimization, which are expanded in Chapters 8 and 9, respectively. In the subsequent chapters, I delve into the real-world challenges these use cases present, propose evaluation approaches that account for both performance and robustness, study trade-offs among them, and analyze the influence of the design in the trade-offs.

## 7.3 Case studies

This section presents the two real-world use cases that support the analysis of the question *how to improve real-world robustness from the domain-specific perspective?*

The first use case corresponds to the problem of frequency assignment or frequency plan design for satellite constellations and the second use case consists of the problem of molecular optimization or drug design; they are covered in Chapters 8 and 9, respectively. In both cases, after identifying trends in the literature that prove the community is not aligned with the deployment of DRL systems in the real world, I examine aspects of how these systems are commonly designed, propose changes for improving robustness, and analyze the trade-offs.

### 7.3.1 Frequency plan design for satellite constellations

The satellite communications landscape has been undergoing significant changes during the last few years. The growth of streaming platforms and other data-intensive services has pushed a traditionally television/video-centered industry into a data-dominated market characterized by higher throughput demands [279]. In addition, new ground stations in unserved communities where terrestrial networks are not an affordable option, as well as new mobile terminals in almost every plane, make user bases larger and more complex. In response, highly-flexible mega constellations are being designed and launched to orbit, and eventually will flood the market in the upcoming years. What once was a static process from a resource control point of view is becoming increasingly dynamic and more challenging.

Satellite operators have been able to incorporate the necessary hardware improvements to adapt to this new dynamic context. Beamforming capabilities capture more complex terrestrial landscapes, digital payloads allows for real time power and bandwidth control [22], and the industry is entering an era marked by mega constellations that add new degrees of freedom [366]. However, current resource management policies are human-driven, which is no longer sufficient given the high-dimensionality and flexibility of upcoming systems. Therefore, operators are looking to outsource many of these human decisions to autonomous agents [1], as it will be the key to being competitive in this growing market [74].

DRL and other AI solutions are prominent in satellite communications research as a promising way to control these complex space systems [113,246]. It has been proven that DRL has the potential to meet the operational constraints that the previously-adopted optimization approaches in the community no longer can [135]. DRL has been studied for the different subproblems that conform the Dynamic Resource Management (DRM) problem (see Figure 7-2): the beam placement and shaping subproblem [180], the routing subproblem [142], the frequency assignment subproblem

[114, 132, 179, 232, 413], and the power allocation subproblem [114, 133, 135, 232, 408]. Overall, it is a time-sensitive problem, as the dynamic behavior of the users' demand requires updating parts of the solution to the subproblems in real time. Given these time constraints and the high-dimensionality context of upcoming constellations, DRL is seen as an essential tool to keep up with the constantly-changing demand in a way that won't require operators to severely over-provision resources.

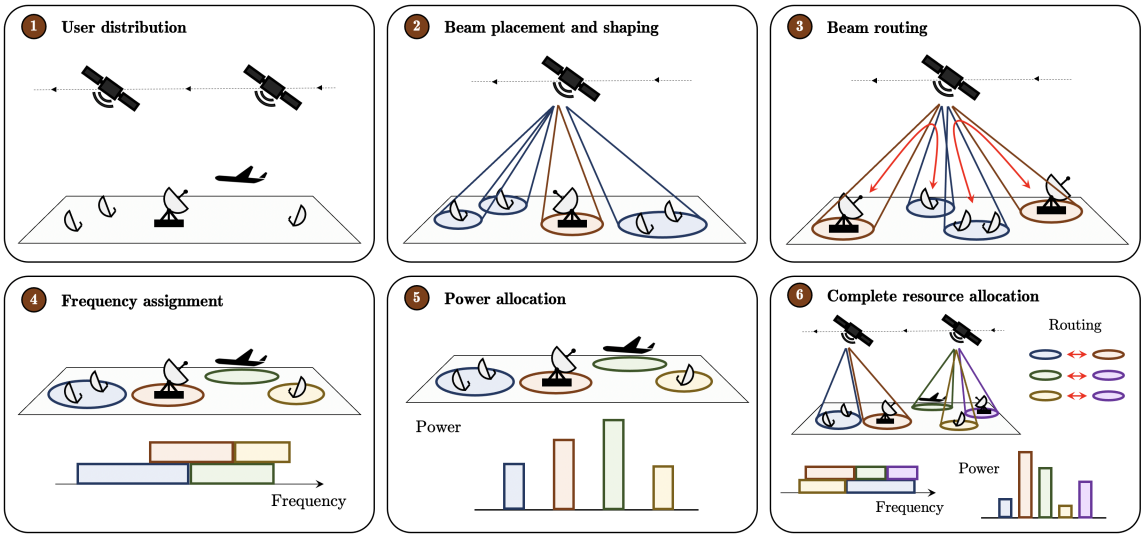



**Figure 7-2:** Dynamic Resource Management problem in satellite communications divided into a sequence of subproblems. From a set of fixed and mobile users with certain demand requirements (step 1), operators first decide how many beams to use, as well as their location and shape (step 2). Then, each beam is routed to a gateway (step 3) and a certain amount of bandwidth within the available frequency spectrum is allocated (step 4). The final subproblem involves powering each beam (step 5), resulting in a complete resource allocation (step 6).

This dissertation specifically focuses on the frequency assignment subproblem (from now on problem); the work is presented in Chapter 8, which is based on [132]. This problem consists of assigning a central frequency and a bandwidth amount to each beam in a satellite constellation [253]. This can be seen as a partitioning problem, although some beams can have access to the same resources given the frequency reuse mechanisms present in the satellite (e.g., different polarizations). In contrast, other beams might not be able to share spectrum since their footprints might be close and therefore interference is present, or because they overlap in the satellite handover schedule. Different versions of this problem have been already addressed by means of DRL [114, 132, 179, 232, 413].

The majority of the literature on DRL for the frequency assignment problem

proves DRL is capable of matching the state-of-the-art performance in the specific task considered, but mostly focus on nominal scenarios, provide little insight on modeling decisions, and fail to address operation considerations that are connected with the domain-agnostic challenges of DRL, such as high-dimensionality and non-stationarity. While the expected dimensionality is around thousands of beams, many papers don't use test cases that go beyond 100 beams. As far as I know, this dissertation presents the first DRL system tested on cases with more than 1,000 beams. In addition, literature generally assumes fixed user bases, while the reality might involve highly-fluctuating and non-stationary user bases. In the context of satellite communications, to the extent of my knowledge, this is the first work testing a DRL model for robustness under non-stationarity.

### 7.3.2 Molecular optimization

The goal of the molecular optimization problem (also referred to as drug design or drug discovery) is to identify novel small molecules with desirable properties and/or activity against a drug target that have the chance to be approved as a drug later in clinical studies. This problem is challenging because the chemical space is huge (between $10^{30}$ and $10^{60}$ different molecules) [341] and there is a large risk of a candidate molecule failing clinical trials. Historically, methods such as genetic algorithms [95] and virtual screening were used to find molecules with desired property profiles. With the advancement of novel generative models in different domains, researchers have been testing many of these new generative methods for the design of novel molecules [97, 104, 195, 196, 239] but the problem of identifying the most promising/optimal compound remains an active area of research.

To try addressing this gap, many studies in the recent years have proposed DRL as a solution to search better over the molecular space [146, 281, 306, 341, 397, 412, 418]. The main benefit of DRL for this problem is the ability to encode property-relevant information into the reward function, this way molecule generation and target profiling can be combined into a single process. However, despite the evident interest in this technology, there is little consensus on which are the most promising models and how DRL should be robustly integrated in industrial processes. The community has not paid enough attention to properly benchmark DRL models for molecular optimization with respect to each other and against previous approaches.

Molecule generation is generally treated a sequential decision-making process, as depicted in Figure 7-3. Generative models work with a specific representation of a

molecule (e.g., graph, SMILES string) and generate a full molecule atom by atom or scaffold by scaffold in an iterative fashion. This process can sometimes take more than 100 steps. A common approach in the literature is to first train a prior generative model from a molecular database (between $10^5$ - $10^8$ different molecules) that simply learns to produce molecules that are valid. In a second stage, the prior is used as the initial policy for a DRL model to finetune, similar to the approach large language models rely on [64, 288]. The goal of the DRL agent is to change the generation distribution so that it favors molecules that optimize specific properties, encoded via the reward function.

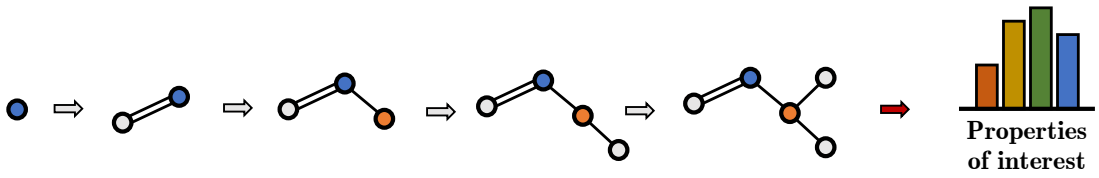



**Figure 7-3:** Process of molecular design and optimization.
Process of molecular design and optimization. A molecule is typically designed sequentially and its properties of interest measured at the end. The goal of molecular optimization is to find molecules that maximize the properties of interest.

This dissertation treats the molecular optimization problem as the second real-world use case in which to further examine how DRL systems are currently designed. While it has been proved that DRL can help find better potential candidate molecules faster, and accelerate the whole synthetization pipeline [412], there is still a lack of understanding which models work better for which kind of subproblems and which are the trade-offs among design decisions. Many of the experimental setups considered in the literature focus on discovering new molecules within distribution to maximize basic synthesizability metrics and/or simple properties. However, in the real world, factors such as more complex atomic structures, out-of-distribution optimization, multiproperty optimization, or non-synthesizable compounds can become hard challenges for DRL. The goal in Chapter 9 is to review the literature, identify the most common design choices, and compare them under the same benchmarks, with increasing levels of complexity in order to test both performance and robustness.

## 7.4 Chapter summary and Contributions

This chapter sets the stage for an in-depth look at domain-specific aspects of DRL robustness, which is the main focus for the rest of this dissertation. As outlined in

Chapter 2, two major hurdles in domain-specific RL research are the lack of diverse use cases beyond robotics and an overemphasis on performance at the expense of robustness. The two real-world applications that follow in this chapter tackle these hurdles, providing insights on the performance versus robustness trade-off for those specific cases.

The chapter has been split into two main sections. The first has dived into the typical workflow that domain-specific RL research follows when developing new DRL solutions for a particular application. In essence, it involves eight high-level steps, starting from identifying the problem and framing it as a Markov Decision Process to evaluating performance and making necessary design changes. It is through this workflow that we have identified three key gaps that could explain why robustness often gets sidelined: 1) Not fully understanding the real-world RL challenges present in the problem, 2) Not defining the subproblems that matter most for robustness, and 3) Not testing the agent across different subproblems to better grasp the performance versus robustness trade-offs.

The second part of this chapter has introduced two real-world use cases, each addressing these gaps in their own way. Chapter 8 will explore frequency assignment for satellite constellations, a problem which has been already studied through a DRL lens but without considering the role of robustness. Here, this dissertation will delve into high-dimensionality and non-stationarity, two critical real-world RL challenges. Conversely, Chapter 9 will take a closer look at the molecular optimization problem, focusing on the challenges of multi-objectiveness and out-of-distribution optimization.

The specific contributions of this chapter are the following:

**Contribution 7.1** Discussed the typical workflow domain-specific RL research follows when proposing new real-world RL methods, as well as the robustness-related gaps that are frequently overlooked.

# Chapter 8

# Case Study 1: Deep Reinforcement Learning for Frequency Plan Design in Satellite Constellations

This chapter follows from the previous chapter and presents an approach to real-world DRL system design for a specific use case: frequency assignment in satellite constellations. The goal of the chapter is to design a DRL system for this problem in which the main design factors and their trade-offs are examined from the perspective of both performance and robustness. To that end, Section **8.1** reintroduces the problem and reviews the related literature. Then, Section **8.2** formalizes the problem, presenting the main variables and restrictions to consider. To gain clarity, these are transformed into an Integer Linear Program, which Section **8.3** focuses on. Next, the main design needs for a DRL system for this problem are discussed in Section **8.4**. Then, the experimental analyses are concentrated in Section **8.5**, which is divided into different questions. The results are later discussed in Section **8.6**. Finally, Section **8.7** concludes the chapter and presents the main contributions.

## 8.1 Introduction

Chapter 7 introduced two real-world use cases which this dissertation tries to find more robust DRL solutions for. One of them is the frequency plan problem, which is addressed in this chapter. As discussed in the previous chapter, DRL is becoming a promising method to solve many of the upcoming challenges in communications, especially in satellite communications [246]. Given the scalability of new constella-

tion systems, the upcoming dynamic and uncertain environments, and the need to exploit decision-making flexibility, making frequency assignment decisions is becoming a complex optimization problem. Operators are looking into solutions like DRL to meet the new set of operational requirements. Although there have been studies proposing DRL to solve aspects of frequency assignment, no real-world deployments have been reported.

One important bottleneck is that, despite the positive results of DRL in the literature, the majority of studies mostly focus on nominal or average-case scenarios, provide little insight on design decisions, and fail to address the operational considerations of deploying DRL systems. Therefore, current DRL systems for frequency assignment in satellite constellations are not robust. Specifically, one can observe two important real-world challenges present in this problem: high-dimensionality and non-stationarity. I argue one reason these challenges are not well-addressed is poor design choices, although they are rarely discussed in the literature (as outlined in Chapter 4, they are usually treated as elements to be reported).

The work in this chapter shifts the target from results and performance to design and robustness. Besides analyzing nominal performance, the experiments in this chapter try to fill the gap that other studies have not addressed in terms of modeling operational requirements. In that sense, constellations with hundreds and thousands of beams are simulated (as opposed to considering individual satellites with less than 50 beams). Similarly, the experiments include non-stationary environments in which, after deployment, the demand distribution changes with respect to the used during training. Relying on models that assume static user distributions could be detrimental during real-time operations and it is seldom considered in the literature. Part of the work in this chapter was originally published in [132].

### 8.1.1 Related work

The frequency assignment problem typically bifurcates into two distinct sub-problems: central frequency assignment (essentially, which central frequency to choose) and bandwidth allocation (essentially, how much bandwidth to allocate). Previous research has proposed solutions to one or both of these sub-problems. The related work on these three different classifications, namely, central frequency assignment, bandwidth allocation, and the combined task of central frequency and bandwidth assignment, along with the experimental setups used in the literature, are summarized in Table 8.1. It can be observed that there is a trade-off between performance and

scalability in the solutions proposed in the literature. More complex algorithms, such as those employing metaheuristics, Linear Programming, or RL, exhibit positive outcomes in simpler scenarios, typically involving a single satellite with a limited number of beams. Conversely, heuristic solutions based on rules are verified in larger-scale situations (such as Non-Geostationary Satellite Orbit constellations with an increased number of beams), but they fail to ensure optimal assignments.

**Table 8.1:** Literature review summary for the frequency assignment problem in satellite communications. Frequency Reuse (FR) refers to whether the proposed solutions allow for all beams to use all available spectrum and polarizations (✓) or the frequency reuse strategy (beam color and polarization) is fixed (e.g., four color frequency reuse). Other acronyms: CF=Central Frequency, BA=Bandwidth Allocation, LP=Linear Programming.

| Problem | Reference | Method | Satellites | Beams | FR |
|---|---|---|---|---|---|
| CF | Camino et al. [49] | Greedy alg., simulated annealing | 1 GEO | 150 | ✓ |
| CF | Kiatmanaroj et al. [202] | LP, greedy alg. | 1 GEO | 200 | – |
| CF | Hu et al. [179] | DRL | 1 GEO | 37 | ✓ |
| BA | Kisseleff et al. [213] | Heuristic | 70 LEO | 347 | – |
| BA | Park et al. [295] | Lagrange multipliers, heuristic | 1 GEO | 20 | – |
| BA | Choi et al. [62] | Water-filling | 1 GEO | 100 | – |
| BA | Wang et al. [164] | Lagrange multipliers | 1 GEO | 10 | – |
| BA | Abdu et al. [5] | Successive Convex Approximation | 1 GEO | 67 | – |
| BA | Paris et al. [293] | Genetic algorithm | 1 GEO | 63 | – |
| BA | Pachler et al. [289] | Particle Swarm Optimization | 1 GEO | 200 | – |
| BA | Ferreira et al. [114] | DRL | 1 LEO | 1 | – |
| BA | Ortiz-Gomez et al. [284] | Convolutional Neural Networks | 1 GEO | 82 | – |
| CF + BA | Kibria et al. [203] | LP | 1 GEO | 8 | ✓ |
| CF + BA | Abdu et al. [4] | LP | 1 GEO | 27 | ✓ |
| CF + BA | Zheng et al. [414] | DRL | 1 LEO | 40 | ✓ |
| CF + BA | Hu et al. [178] | DRL | 1 GEO | 37 | ✓ |
| CF + BA | Salcedo et al. [325] | Hopfield neural network, genetic alg. | 1 GEO | 200 | – |
| CF + BA | Cocco et al. [71] | Simulated annealing | 1 GEO | 200 | ✓ |
| CF + BA | Pachler et al. [291] | Greedy algorithm | 4,400 LEO | 10,000 | ✓ |

**Central Frequency Assignment problem** Camino et al. [49] represented this problem as a graph coloring problem, using local search and Simulated Annealing (SA) as their approach. Kiatmanaroj et al. [202] noted that traditional Integer Linear Programming (ILP) optimizers face scalability limitations and hence put forward a greedy algorithm as a solution. Some researchers turned to AI algorithms to meet operational demands. For instance, Hu et al. proposed a DRL model to tackle the Dynamic Channel Allocation (DCA) problem. The performance of this model

was found to align closely with that of leading-edge DCA algorithms. However, it's important to note that these studies were conducted in settings involving fewer than 200 beams. The scalability of these approaches becomes more constrained when frequency reuse (FR) variables are incorporated into the optimization process [49,179].

**Bandwidth Allocation problem** Kisseleff et al. [213] looked into bandwidth allocation in a LEO constellation with 70 satellites and 350 beams. However, the solution they offered was limited to discovering a viable heuristic allocation. Park et al. [295] put forth a binary search-based method, a more equitable alternative to the water-filling approach that Choi et al. [62] had previously considered. Subsequent research [5, 164] also utilized the same fairness-based optimization objective but implemented convex optimization to allocate bandwidth. Both of these studies assessed their methodologies on single-satellite and single-gateway cases with no more than 70 beams. Bandwidth allocation in single-satellite systems also saw the use of metaheuristics [290, 293] and AI techniques [114, 284]. In all the cited studies, other variables like power or the roll-off factor were taken into account for simultaneous optimization. However, the effectiveness and robustness of these methods were not evaluated for systems with over 200 beams.

**Joint Central Frequency Assignment and Bandwidth Allocation** Both conventional optimization and AI methods were proposed for the joint problem, with the majority of the studies also accounting for FR optimization. Some of the studies proposed Linear Programming. For example, Kibria et al. [204] addressed multi-user aggregation and access control design for carrier aggregation, while Abdu et al. [4] incorporated power optimization as well. Both works were tested in a single GEO satellite setup with fewer than 30 beams. DRL has also been considered as a potential solution. This was explored by Zheng et al. [414] and Hu et al. [178], who highlighted that traditional optimization methods place a severe limit on use cases with time constraints. Both evaluated their solutions in scenarios involving a single satellite and up to 40 beams. Salcedo-Sanz et al. [325] utilized a neural network combined with a genetic algorithm to tackle the problem from an interference minimization perspective, testing it on multiple scenarios with a maximum of 36 beams and 128 bandwidth channels. The same issue was examined by Cocco et al. [71], who also optimized for power and used a capacity-oriented objective function and the SA algorithm. Their tests on a single satellite with 200 beams and 16 bandwidth channels demonstrated that this method reduces both unmet and excess capacity compared to traditional

approaches using conventional payloads. The constraint satisfaction algorithm presented by Pachler et al. [291] is the only identified solution that addresses scenarios involving more than 500 beams. The aim of using DRL is to further decrease the time that greedy approaches take to solve the problem.

## 8.2 Problem statement

The goal is to design a *frequency plan* for a constellation with $N_S$ identical multibeam satellites, all situated in the same orbital plane as depicted in Figure 8-1. This necessitates entirely outlining the frequency utilization for all of the constellation's $N_B$ beams. Each beam can serve one or more users, or gateways that connect to the ground segment. It is assumed that all beams consistently serve their designated gateway or user group, which could be mobile —in such cases, the beam "tracks" the users. An essential input to this problem is the throughput required by each user, which could correspond to fluctuating demands or committed rates defined by user contracts. At any given moment, each beam draws power from a single satellite within the constellation. For Non-Geostationary Orbit (NGSO) satellites, handover operations occur, resulting in changes over time in the satellite powering each beam.

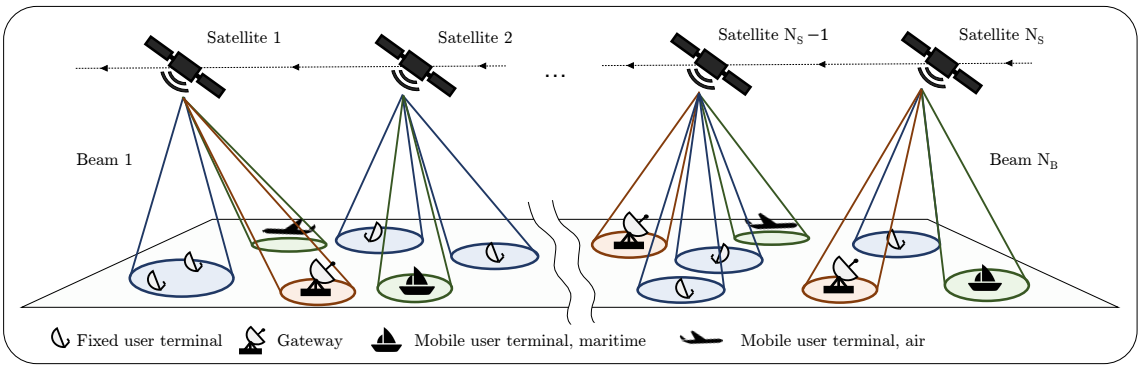



**Figure 8-1:** A constellation with $N_S$ identical satellites in the same orbit and $N_B$ beams is considered. Gateways, fixed terminals, and mobile users are connected to the network.

When it comes to frequency resources, all satellites are permitted to use the same specific portion of the spectrum, which is partitioned into $N_{BW}$ equivalent bandwidth channels or slots. Similarly, all satellites possess the same frequency reuse capabilities: there are $N_{FR}$ frequency reuses accessible, along with $N_P$ polarizations for each reuse. Polarizations enable the utilization of more spectrum in a condensed area

without causing additional interference. For instance, when using right-handed and left-handed circular polarizations, $N_P$ equals 2.

To define a full frequency plan, the operator must decide, for each beam, how many and which bandwidth slots are assigned, and which reuse group and polarization should be used. Formally, this corresponds to selecting, for every beam $b \in \{1, ..., N_B\}$:

- A discrete number of consecutive bandwidth slots $b_b$, which cannot be greater than $N_{BW}$. This number might have a lower bound required to satisfy the link budget equation.
- A positive integer $f_b$, that indicates the first bandwidth slot used. Beam $b$ then uses slots $f_b, f_b + 1, ..., f_b + b_b - 1$, all of them part of the available spectrum.
- A positive integer $k_b$ representing a frequency reuse out of the $N_{FR}$ available.
- In case $N_P = 2$, a binary variable representing the chosen polarization.

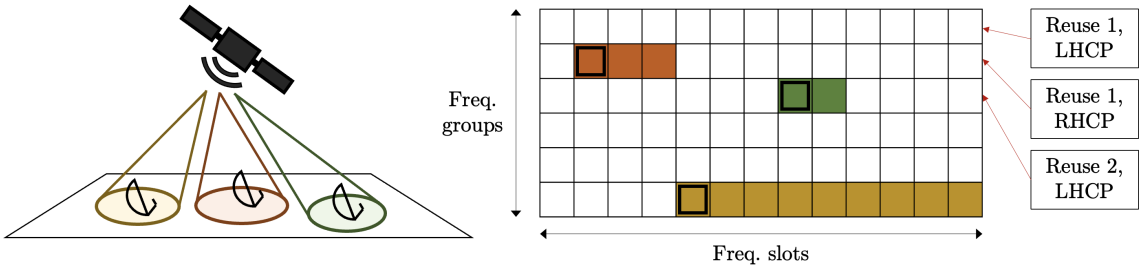



**Figure 8-2:** Frequency assignment representation in grid form with $N_{FR} \cdot N_P$ rows and $N_{BW}$ columns. In this example, $N_{FR} = 3$, $N_P = 2$, and $N_{BW} = 13$. Each of the 3 beams being powered by the satellite is assigned to a cell in the grid representing the first slot (black squares) and to a certain number of consecutive slots (colored cells). For example, the beam depicted in green is assigned to reuse group 2, left polarization, and is using slots 8 and 9.

Figure 8-2 presents a graphical representation of this decision space structured as a grid, with $N_{FR} \cdot N_P$ rows and $N_{BW}$ columns. Each column represents a bandwidth slot, while each row corresponds to a combination of a frequency reuse and a polarization. Rows are arranged first by frequency reuse and then by polarization. With this representation, assigning a frequency for a beam translates into selecting a specific cell in the grid, corresponding to the first slot (depicted as black squares in the figure) and choosing a valid number of slots (represented as colored cells).

The constellation's orbit is assumed to potentially be NGSO. As a result, frequency plans need to consider handover operations. While new digital payloads will

enable the reassignment of frequency resources during a handover, it may not always be possible or preferred due to various reasons. In such scenarios, beams continue to utilize the same frequency resources when transitioning between satellites. This scenario is illustrated in Figure 8-3, where the green beam switches from satellite 1 to satellite 2, maintaining the same frequency reuse, polarization, and bandwidth slots.

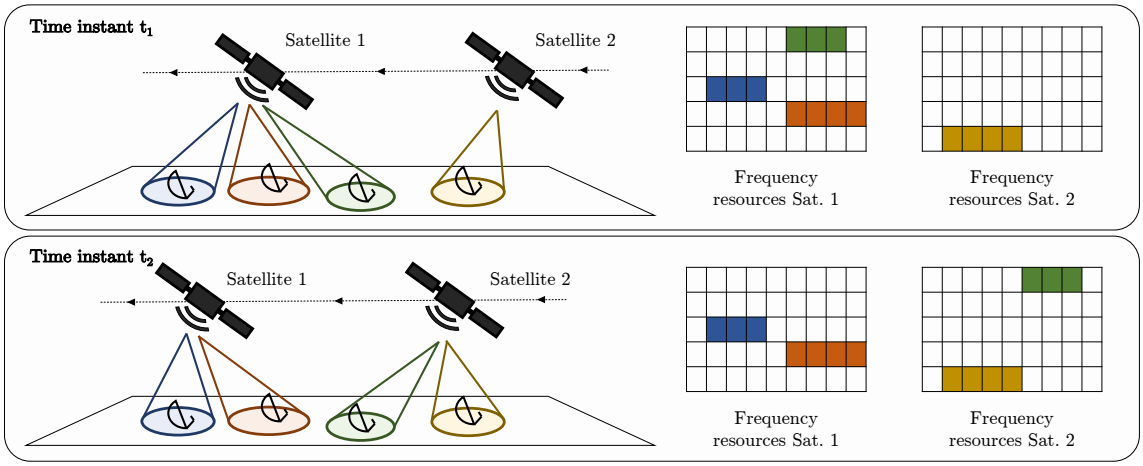



**Figure 8-3:** Handover operation example between two satellites at time instants $t_1$ and $t_2$.

During a handover, it's crucial that the resources slated for use on the incoming satellite aren't already being used by another beam. This consideration forms a significant constraint and applies to any pair of beams powered by the same satellite at any moment. We identify this constraint for a pair of beams as an *intra-group* restriction between these beams. Therefore, the set $\mathcal{R}_A$ encompasses all pairs $(i, j)$ where beams $i$ and $j$ maintain an intra-group restriction. We operate under the assumption that this set is determined beforehand and is externally updated during operation as required.

A second type of restriction involves potential interference if two beams with closely placed footprints utilize the same polarization and their designated bandwidth slots overlap. This situation is termed an *inter-group* restriction, as depicted in Figure 8-4. Like in the case of the handover constraint, the set $\mathcal{R}_E$ encodes all pairs of beams $(i, j)$ sustaining an inter-group restriction. Different interference criteria could be used to construct this set, such as SINR level or angular distance threshold. We presume that the specific choice, and consequently the set $\mathcal{R}_E$, are provided by the operator.

The final type of restriction pertains to gateway dimensioning. The operator may desire to reduce bandwidth use for a pair of beams, not just when they overlap in the handover schedule or when they could potentially interfere with each other, but

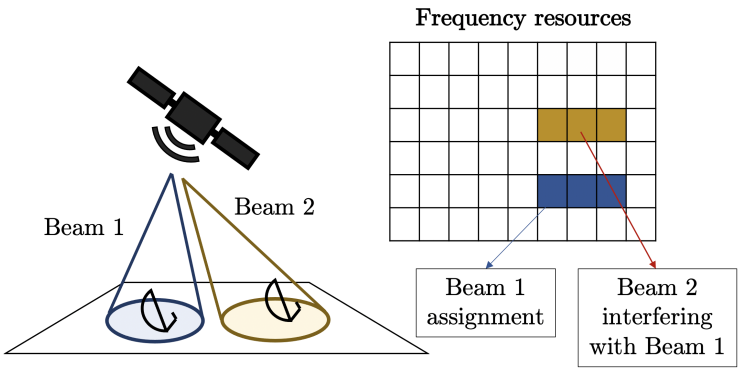



**Figure 8-4:** Inter-group restriction example between beam 1 and beam 2. Diagram shows the moment beam 2 is to be assigned and beam 1 has already been assigned.

also if they utilize resources from the same gateway at any given time. The objective is to design the frequency plan in a way that accounts for the gateway dimensioning as well. We assume that the usable spectrum at the gateway aligns with the usable spectrum at the satellite. Furthermore, the gateway employs the same number of polarizations, $N_P$, but cannot reuse frequency. We also have access to the set $\mathcal{R}_G$, which encodes all pairs of beams $(i, j)$ that share a gateway at any point in time.

Ultimately, the optimization scope of this problem formulation is frequency assignment at the beam level, a problem which, to the best of my knowledge, remains unresolved in high-dimensional, dynamic constellations with frequency reuse mechanisms. A further layer of decomposition could potentially involve splitting the beams into multiple carriers to serve each user. However, frequency assignment at the carrier level falls outside the scope of this dissertation. This issue can be addressed in various ways that can coexist with the optimization framework (for example, TDMA, FDMA, or CDMA). The methods introduced in this chapter, by considering interference among beams, ensure that carriers within one beam do not interfere with carriers from other beams. Therefore, frequency assignment at the carrier level could be solved locally for each beam.

## 8.3 Integer Linear Programming formulation

To provide a mathematical grounding for this problem before designing a DRL system to solve it, this section presents an Intener Linear Programming (ILP) formulation that encodes each decision and restriction as variables and constraints, respectively. While this is an adequate method to easily encode problem needs, constraints, and objectives with low granularity with respect to the decision variables and only requires

a commercial solver, it is not well-suited for high-dimensional scenarios. In contrast, the advantage of DRL is that decision-making consists of forward-passes of a neural network.

The following lines outline how different features of the frequency assignment problem are encoded in the ILP formulation using linear operators. It is assumed that constraint sets $\mathcal{R}_A$, $\mathcal{R}_E$, and $\mathcal{R}_G$ are an input. The formulation is framed for optimizing the downlink frequency plan, although it is also compatible with uplink frequency plan design.

**Frequency plan decisions** On the beam level, frequency assignment decisions consist of choosing how many bandwidth slots, which ones, and which frequency reuse and polarization to use. From the perspective of the grid representation introduced in Figure 8-2, this means selecting 1) a column and 2) a row in the grid, and then 3) a number of consecutive slots. The formulation encodes the column (i.e., the first slot) as an integer variable $f_i$, with domain $\{1, ..., N_{BW}\}$, for each beam $i \in \{1, ..., N_B\}$. Then, the row (i.e., frequency reuse and polarization) is encoded as an integer variable $g_i$, with domain $\{1, ..., N_{FR} \cdot N_P\}$. Finally, the number of consecutive slots is encoded as an integer variable $b_i$, with the same domain as $f_i$. These variables are formally defined as follows:

$$f_i \in \{1, ..., N_{BW}\}, \qquad \forall\, i \in \{1, ..., N_B\} \tag{8.1}$$
$$g_i \in \{1, ..., N_{FR} \cdot N_P\}, \ \forall\, i \in \{1, ..., N_B\} \tag{8.2}$$
$$b_i \in \{1, ..., N_{BW}\}, \qquad \forall\, i \in \{1, ..., N_B\} \tag{8.3}$$

Any frequency plan can then be decoded using these three variables per beam. Any optimization method should return these values to the operator.

The bandwidth variable (8.3) has been defined with domain $\{1, ..., N_{BW}\}$. However, the operator might be interested in specifying higher lower bounds, for contractual reasons or since using a single bandwidth slot might not be enough to satisfy the link budget equation for certain beams [253]. This way, variable (8.3) could be redefined as

$$b_i \in \{c_i, ..., N_{BW}\}, \ \forall\, i \in \{1, ..., N_B\} \tag{8.3}$$

where $c_i$ is the minimum number of slots that beam $i$ requires. Similarly, the domain of variables (8.1) and (8.2) could also be changed to split frequency resources following criteria specified by the operator or the users' contracts. This is left outside of the

scope of this dissertation.

**Constraint: Limited spectrum** The focus now shifts to the constraints of the problem. First, we encode the constraint that all bandwidth slots used must be within the spectrum limits imposed by the system (i.e., only the $N_{BW}$ considered slots can be used). This is encoded as follows:

$$f_i + b_i - 1 \leq N_{BW}, \;\; \forall i \in \{1, ..., N_B\} \tag{8.4}$$

**Constraint: Intra-group or handover restrictions** The next step is accounting for the intra-group restrictions, given by the set $\mathcal{R}_A$. This type of restrictions are caused by handover operations and are relevant if and only if constrained beams that use the same frequency reuse and polarization (i.e., same $g_i$) overlap in their bandwidth slots. I first introduce the constraints that encode the intra-group restrictions and then describe each of the elements involved. A pair of constraints is defined per restriction:

$$f_i + b_i \leq f_j + N_{BW}(2 - y_{ij} - z_{ij}), \;\; \forall\, i, j \text{ s.t. } (i, j) \in \mathcal{R}_A \tag{8.5}$$
$$f_j + b_j \leq f_i + N_{BW}(1 - y_{ij} + z_{ij}), \;\; \forall\, i, j \text{ s.t. } (i, j) \in \mathcal{R}_A \tag{8.6}$$

Both constraints make use of two types of auxiliary binary variables: $y_{ij}$ and $z_{ij}$, which are now explained. Constraints (8.5) and (8.6) enforce a non-overlapping frequency assignment between beams $i$ and $j$ holding an intra-group restriction if and only if both beams use the same reuse group and polarization. To ignore these constraints in case they do not, $N_{BW}$ is added to the right-hand side of the inequalities, as $f_i + b_i$ is upper-bounded by $N_{BW}$. To that end, variable $y_{ij}$ is used and the following constraints enforced:

$$y_{ij} \in \{0, 1\}, \qquad \forall\, i, j \text{ s.t. } (i, j) \in \mathcal{R}_A \tag{8.7}$$
$$g_i \geq g_j - N_{FR} \cdot N_P(1 - y_{ij}), \;\; \forall\, i, j \text{ s.t. } (i, j) \in \mathcal{R}_A \tag{8.8}$$
$$g_i \leq g_j + N_{FR} \cdot N_P(1 - y_{ij}), \;\; \forall\, i, j \text{ s.t. } (i, j) \in \mathcal{R}_A \tag{8.9}$$

If $y_{ij} = 1$, the formulation enforces that $g_i = g_j$, since both (8.8) and (8.9) are active. If $y_{ij} = 0$, the opposite should occur and these constraints should be ignored, to that end the upper-bound term $N_{FR} \cdot N_P$ is added to the right-hand side of the inequalities. However, enforcing $g_i \neq g_j$ cannot be achieved solely with variable $y_{ij}$. To enforce strict inequality when $y_{ij} = 0$, the formulation introduces binary variables $p_{ij}$ and the

following constraints to account for both cases $g_i > g_j$ and $g_j > g_i$:

$$p_{ij} \in \{0,1\}, \qquad \forall\, i,j \text{ s.t. } (i,j) \in \mathcal{R}_A \tag{8.10}$$
$$g_i - g_j \geq \epsilon - N_{FR} \cdot N_P(1 - p_{ij} + y_{ij}), \quad \forall\, i,j \text{ s.t. } (i,j) \in \mathcal{R}_A \tag{8.11}$$
$$g_i - g_j \leq -\epsilon + N_{FR} \cdot N_P(p_{ij} + y_{ij}), \quad \forall\, i,j \text{ s.t. } (i,j) \in \mathcal{R}_A \tag{8.12}$$

which are active if and only if $y_{ij} = 0$. In that case, and when $p_{ij} = 1$, (8.11) is active and $g_i > g_j$. On the contrary, if $p_{ij} = 0$, (8.12) is active and $g_j > g_i$ holds. When $y_{ij} = 1$, the difference between $g_i$ and $g_j$ is upper-bounded by $N_{FR} \cdot N_P$.

The effect of binary variable $z_{ij}$ comes into play if beams $i$ and $j$ hold a restriction and are assigned to the same frequency reuse and polarization (i.e., $g_i = g_j$). In this case, we want to make sure that $f_i + b_i \leq f_j$ or $f_j + b_j \leq f_i$, i.e., we want to ensure that beam $i$ has allocated spectrum either to the left or to the right of beam $j$, but without overlapping. These two possible scenarios are taken into account with variable $z_{ij}$ and the following constraints:

$$z_{ij} \in \{0,1\}, \qquad \forall\, i,j \text{ s.t. } (i,j) \in \mathcal{R} \tag{8.13}$$
$$f_j - f_i \geq 0 - N_{BW}(1 - z_{ij}), \quad \forall\, i,j \text{ s.t. } (i,j) \in \mathcal{R} \tag{8.14}$$
$$f_i - f_j \geq \epsilon - N_{BW} z_{ij}, \quad \forall\, i,j \text{ s.t. } (i,j) \in \mathcal{R} \tag{8.15}$$

where $\epsilon$ is a very small positive number, and $\mathcal{R}$ represents the union $\mathcal{R}_A \cup \mathcal{R}_E \cup \mathcal{R}_G$ (these constraints are defined for all intra-group, inter-group, and gateway restrictions). Given a restriction $(i,j)$, if $z_{ij} = 1$, (8.14) is active and enforces that $f_j \geq f_i$ (i.e., beam $j$ cannot use lower frequencies than beam $i$'s). On the contrary, if $z_{ij} = 0$, (8.15) is active and the effect is the opposite. These constraints become inactive by leveraging the $N_{BW}$ term on the right-hand side of the inequalities, as $-N_{BW}$ is the lowest value the difference can take.

Note that at most one of the intra-group constraints (8.5) and (8.6) can be active at a given time. Constraint (8.5) does so for the case in which $z_{ij} = 1$ ($f_i \leq f_j$), whereas constraint (8.6) is active when $z_{ij} = 0$ ($f_i > f_j$). To summarize the effect of the auxiliary variables introduced so far, Table 8.2 shows how the different frequency assignment cases between two beams holding an intra-group restriction are encoded by means of variables $z_{ij}$, $y_{ij}$, and $p_{ij}$.

**Constraint: Inter-group or interference restrictions** The inter-group restrictions are given by set $\mathcal{R}_E$ and concern all pairs of beams with close footprints, which might interfere with each other during operations. Setting a threshold for how close

**Table 8.2:** Encoding of different frequency assignment cases by means of auxiliary variables when beams $i$ and $j$ share a constraint.

| Auxiliary variables | Encoded frequency assignment case | Necessary for |
|---|---|---|
| $z_{ij} = 1$ | $f_i \leq f_j$ | Any type of |
| $z_{ij} = 0$ | $f_i > f_j$ | constraint |
| $y_{ij} = 1$ | $g_i = g_j$ (same freq. reuse and polarization) | Intra-group |
| $y_{ij} = 0$ and $p_{ij} = 1$ | $g_i > g_j$ | constraints |
| $y_{ij} = 0$ and $p_{ij} = 0$ | $g_i < g_j$ | (handover) |
| $s_{ij} = 0$ | $m_i = m_j$ (same polarization) | Inter-group |
| $s_{ij} = 1$ and $d_{ij} = 0$ | $m_i < m_j$ | constraints |
| $s_{ij} = 1$ and $d_{ij} = 1$ | $m_i > m_j$ | (interference, gateways) |

two interfering beams can be is up to the operator's policy, which might prefer to trade additional inter-group restrictions for further interference mitigation. One way to define this set might be to impose an angular distance threshold between beams; this decision is left out of the scope of the formulation.

As in the intra-group case, a pair of constraints is defined per restriction:

$$f_i + b_i \leq f_j + N_{BW}(1 + s_{ij} - z_{ij}), \ \forall\, i, j \text{ s.t. } (i,j) \in \mathcal{R}_E \tag{8.16}$$
$$f_j + b_j \leq f_i + N_{BW}(s_{ij} + z_{ij}), \qquad \forall\, i, j \text{ s.t. } (i,j) \in \mathcal{R}_E \tag{8.17}$$

Inter-group constraints rely on auxiliary binary variables $s_{ij}$ and $z_{ij}$. The effect of variable $z_{ij}$ is given by constraints (8.14) and (8.15), which have been introduced previously, and considers the cases in which beam $i$'s spectrum is to the left or right of beam $j$'s. The definition of $s_{ij}$ variables is addressed next.

As introduced in Figure 8-4, inter-group restrictions can negatively impact the performance of the system when both beams are using the same polarization, regardless of their frequency reuse. To specifically focus on polarization, first the following variables and constraints are introduced:

$$k_i \in \{1, ..., N_{FR}\}, \ \ \forall\, i \in \{1, ..., N_B\} \tag{8.18}$$
$$m_i \in \{0, N_P - 1\}, \ \forall\, i \in \{1, ..., N_B\} \tag{8.19}$$
$$g_i = N_P k_i - m_i, \quad \forall\, i \in \{1, ..., N_B\} \tag{8.20}$$

Variable $k_i$ encodes the frequency reuse assigned to beam $i$ whereas variable $m_i$ encodes its polarization.

Similar to constraints (8.7) - (8.9), binary variable $s_{ij}$ is used for each pair of beams holding an inter-group constraint. This variable encodes whether beams $i$ and

$j$ use the same polarization, by means of the following constraints:

$$s_{ij} \in \{0, N_P - 1\}, \qquad \forall\, i, j \text{ s.t. } (i,j) \in \mathcal{R}_E \tag{8.21}$$
$$m_i \geq m_j - (N_P - 1)s_{ij}, \quad \forall\, i, j \text{ s.t. } (i,j) \in \mathcal{R}_E \tag{8.22}$$
$$m_i \leq m_j + (N_P - 1)s_{ij}, \quad \forall\, i, j \text{ s.t. } (i,j) \in \mathcal{R}_E \tag{8.23}$$

If $s_{ij} = 0$ then both (8.22) and (8.23) are active, and $m_i$ and $m_j$ are enforced to be equal. Note that in case $N_P = 1$, then $s_{ij}$ is always zero and $m_i = m_j$, since there is only one polarization.

Then, following the same idea behind constraints (8.10) - (8.12), binary variable $d_{ij}$ is used to help encoding whether $m_i > m_j$ or $m_j > m_i$, i.e., enforcing $m_i$ and $m_j$ to be different in case there is more than one polarization and $s_{ij} = 1$. The following constraints explain this idea:

$$d_{ij} \in \{0, 1\}, \qquad \forall\, i, j \text{ s.t. } (i,j) \in \mathcal{R}_E \tag{8.24}$$
$$m_i - m_j \leq -\epsilon + (N_P - 1)(1 + d_{ij} - s_{ij}), \quad \forall\, i, j \text{ s.t. } (i,j) \in \mathcal{R}_E \tag{8.25}$$
$$m_i - m_j \geq \epsilon - (N_P - 1)(2 - d_{ij} + s_{ij}), \quad \forall\, i, j \text{ s.t. } (i,j) \in \mathcal{R}_E \tag{8.26}$$

which can only be active if $s_{ij} = 1$ (more than one polarization). Then, if $d_{ij} = 0$, (8.25) is active and therefore $m_j > m_i$. On the other hand, if $d_{ij} = 1$, (8.26) is active and $m_i > m_j$. In all cases, the difference $|m_i - m_j|$ is bounded by $N_P - 1$. Table 8.2 also summarizes all cases that inter-group restrictions-related auxiliary variables encode.

**Constraint: Gateway dimensioning** The gateway dimensioning constraints are given by the set $\mathcal{R}_G$ and correspond to beams that cannot overlap in frequency when they connect to the same gateway. Since it is assumed that gateways do not reuse frequency but share the same number of polarizations with the satellites, these constraints can be encoded by replicating the constraints introduced for inter-group restrictions, i.e., constraints (8.16) and (8.17) and auxiliary constraints (8.18)-(8.26). Some pairs of beams $(i, j)$ might be in both $\mathcal{R}_E$ and $\mathcal{R}_G$, in that case the formulation only needs to define the constraints once.

**Other considerations: objective function and computational complexity** While the previous variables and constraints are enough to define feasible frequency plans, and therefore they are enough to understand the feasibility of any DRL solution, an ILP commercial solver will not be able to set priorities among feasible plans. To

that end, an objective function must be defined. There are different options on what to prioritize, all of them can be encoded in a parametrized function. This is discussed in depth in Appendix D.1, since this chapter mostly focuses on feasibility.

The main advantage of DRL over the ILP formulation is runtime, especially when the number of beams $N_B$ is large. Appendix D.2 shows that this problem is NP-hard and possesses a search space of the order $\mathcal{O}((N_{BW}^2 N_P N_{FR})^{N_B})$. This explains why many approaches proposed in the literature might have been limited to experimental setups with only dozens of beams.

## 8.4 Designing a Deep Reinforcement Learning System

This section covers the design of a DRL system to solve the frequency assignment problem for a set of $N_B$ beams of a constellation. The goal is to understand which are the empirical design choices that benefit the performance and robustness of the DRL system in the context of this task. To that end, six different elements of the design process are taken into account: the state representation, the action space, the reward function, the policy network, the policy optimization algorithm, and the training procedure. For each of these elements, different choices are proposed, with the goal of finding the best ones to solve the problem and further understanding the trade-offs between performance and robustness in satellite communications.

Table 8.3 summarizes the variations that are considered for each of the six elements of the DRL system that are being designed. This section discusses the variations for the first three (state, action, and reward), which account for domain expertise. The rest are taken from the literature and will be introduced alongside the experimental setup in the next chapter.

### 8.4.1 Episodes and timesteps

As presented in the previous section, the total number of plans is in the order of $(N_{BW}^2 N_P N_{FR})^{N_B}$, which makes the problem challenging for large values of $N_{BW}$, $N_{FR}$, and $N_B$. While some methods will take a time penalty to compute a solution (e.g., ILP), DRL might still be relatively fast and in many cases able to still operate under time constraints. However, one can assume that the complexity of training and the quality of the solutions obtained will degrade as the dimensionality of the

**Table 8.3:** Main components of the DRL system to be designed and their considered variations.

| DRL system component | Variations considered |
|---|---|
| State representation | *Lookahead* and *without* lookahead |
| Action space | *Grid* and *tetris-like* |
| Reward function | *Each*, *final*, and *Monte-Carlo* |
| Policy network | CNN+MLP and CNN+LSTM |
| Policy optimization algorithm | DQN and PPO |
| Training procedure | Same as test and harder than test |

problem increases. To that end, the design of episodes and timesteps considered for the problem will take scalability into account.

Specifically, a timestep is defined as the frequency assignment for just one beam. Then, the episode is defined as the complete frequency assignment of all $N_B$ beams. By doing this, the dimensionality of the problem reduces to $N_{BW}^2 N_P N_{FR}$ for each timestep. Since the goal is to factor in robustness into the analysis, the problem is simplified to select only the first bandwidth slot, the frequency reuse, and the polarization, therefore the total number of used slots is given (i.e., according to the formulation, for each beam $b_i$ is given and the agent picks $f_i$ and $g_i$). Specifically, for each beam, it is assumed the problem provides the minimum number of slots necessary, this corresponds to $c_i$ in equation (8.3). Therefore, the problem is reduced to pick a cell in the grid representation depicted in Figure 8-2, with $N_{BW} N_P N_{FR}$ different options.

### 8.4.2 Action space

Two different action spaces are studied, these are pictured in Figure 8-5, which shows a scenario/grid with $N_P N_{FR} = 4$ and $N_{BW} = 4$. One alternative is to directly choose a cell in the grid as the action. This action space, which is defined as *grid*, consists of $N_{BW} N_P N_{FR}$ different actions. The second action space is defined as *tetris-like* and only contemplates five possible actions. In this space, for each beam, a random frequency assignment is first made for a beam, i.e., a cell in the grid is randomly chosen. Then, the agent is able to move it up, down, left, and right across the grid until the fifth action *new* is chosen and a new beam undergoes the same procedure. Note that with this latter approach, episodes take a longer and random number of timesteps, since one beam can take more than one timestep to be assigned and there

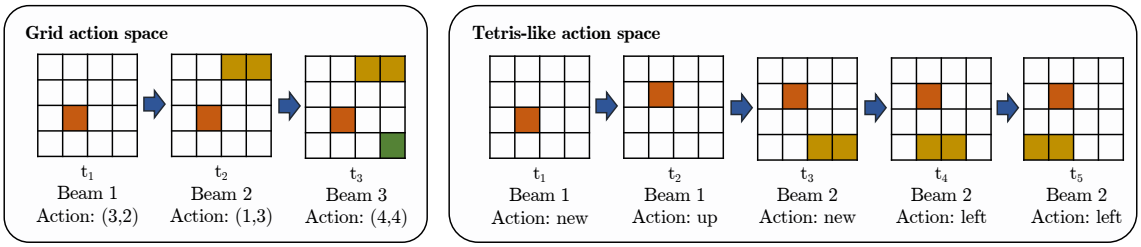



**Figure 8-5:** Two action spaces considered for the frequency assignment problem: *grid* and *tetris-like*.

is no restriction on how many intermediate actions should be taken before taking action *new*. The advantage of this representation is the substantial reduction of the action space, which benefits the learning algorithm.

### 8.4.3 State representation

Next, two different state representations are considered. In both of them, the state is defined as a 3-dimensional tensor in which the first dimension size is 1, 2, or 3, and $N_P N_{FR}$ and $N_{BW}$ are the sizes of the second and third dimensions, respectively. A slice of such tensor along the first dimension is referred to as a *layer*. To better understand both representations, let's consider a certain timestep, in which the assignment for one specific beam $k$ is being made, $k-1$ beams have already been assigned, and there are $N_B - k$ to go. Below is a description of each layer:

1. In both representations the first layer (with dimensions $N_P N_{FR} \times N_{FS}$) stores which grid cells conflict with beam $k$, since they are "occupied" by at least one of the beams, among the $k-1$ already assigned, that have some kind of constraint with $k$.
2. In the case the action space is *tetris-like*, the second layer stores the current assignment of beam $k$ – this is done regardless of the state representation chosen.
3. The last layer serves as a *lookahead* layer and is optional. This layer contains information regarding the remaining beams, such as the number $N_B - k$, or the amount of bandwidth that will be compromised in the future for beam $k$ due to some of the beams remaining to be assigned having an intra-group or inter-group constraint with beam $k$.

Figure 8-6 shows an example of what each of these three layers might look like. Whether the last layer (lookahead layer) is included in the tensor or not defines the

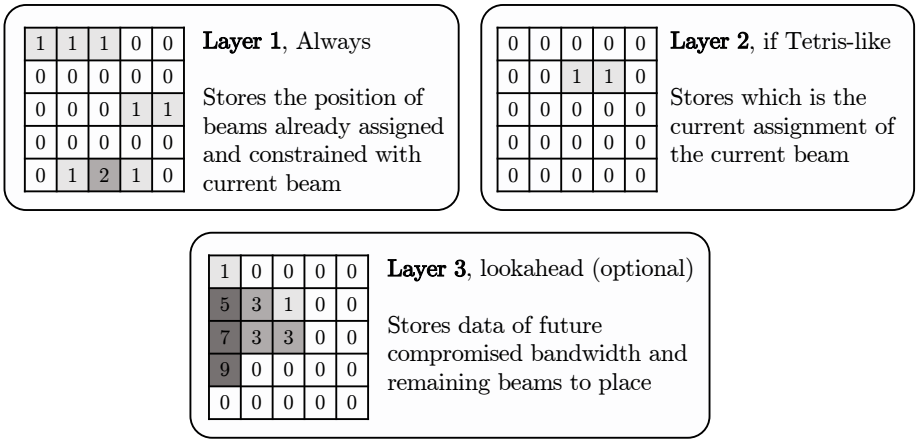



**Figure 8-6:** Different possible layers of the state space.

two state representations considered. Training runs will use state representations *with* or *without* the lookahead layer.

Note that the selection of state and action representation entails an additional benefit: the agent can start from incomplete frequency plans. There might be situations in which the operator might not want to entirely reconfigure a plan but just reallocate a reduced set of beams (e.g., due to moving users, handover reconfigurations, etc.). Furthermore, adding or removing beams to the constellation would not be a problem either. In that sense, the model would be adapt to changes in the beam placement.

### 8.4.4 Reward function

Then, three alternative functions are considered to define the **reward**. For the three of them, if the action space is *tetris-like*, the reward function is only applied whenever the action *new* is chosen, otherwise the value $\frac{-1}{N_{FG} \cdot N_{FS}}$ is given as a reward. This is done to avoid the agent finding local optima that consist of just moving a specific beam around without calling the action *new*.

To help defining the three reward functions, let's denote by $B(s_t)$ the number of successfully already-assigned beams at state $s_t$. A beam is successfully assigned if it does not violate any constraint. The rewards are then computed as:

1. **Each**: Once a beam is assigned (i.e., any action is taken in the *grid* action space or the action *new* is taken in the *tetris-like* action space), the first reward function consists of computing the difference in the number of successfully already-assigned beams between the states at the current timestep and previous

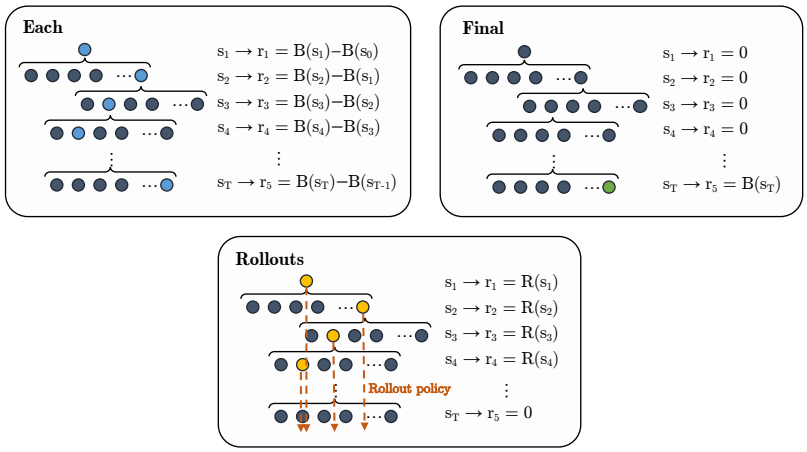



**Figure 8-7:** Reward function choices considered. $B(s_t)$ corresponds to the number of successfully already-assigned beams at state $s_t$.

timestep, i.e., $r_t = B(s_t) - B(s_{t-1})$. Note this number can be negative.

2. **Final**: The second alternative is to only compute the final quantity of beams that are successfully assigned once the final state $s_T$ is reached and give a reward of zero at all timesteps before that. Therefore, $r_t = B(s_t)$ if $s_t = s_T$, otherwise zero.
3. **Rollout**: Finally, the third reward function uses a rollout policy that randomly assigns the remaining beams at each timestep. This is only done to compute the reward, the outcome of the rollout policy is not taken as agent's actions. The reward equation is $r_t = R(s_t) = B(s_T)|_{\text{rollout}} - B(s_t)$. Note that $r_T = R(s_T) = 0$.

Each of these strategies is defined as *each*, *final*, and *rollout*, respectively. Figure 8-7 shows a visual comparison between the three options considered.

## 8.5 Results

This section introduces the experimental results of comparing different designs of the DRL system for the frequency assignment problem in satellite communications. The distribution of experiments is as follows: first, Section 8.5.1 presents the experimental setup that is used across experiments, including the constellation models and the data considered. Then, Section 8.5.2 provides an exhaustive comparison across the different combinations of designs when considering the action space, the state representation, the reward function, the training procedure, and the discount factor $\gamma$. Next, Section 8.5.3 focuses on scalability, the first real-world challenge considered for this problem, and provides an updated comparison when scaling the dimensionality

of the constellation model by 5. Section 8.5.4 continues the analysis by incorporating the policy network and the policy optimization algorithm into the design process and comparing different combinations for them. Scalability is further analyzed in the same experiment. Lastly, Section 8.5.5 addresses non-stationarity in the environment, which constitutes the second real-world RL challenge considered for this use case.

### 8.5.1 Experimental setup

To test the proposed DRL system, this section considers a baseline scenario with 100 beams, 7 satellites, $N_P = 2$, $N_{FR} = 2$, and $N_{BW} = 20$. This matches the dimensionality of typical experimental setups found in the literature. From a set of 5,000 beams based on real data provided by SES S.A., a random train and test datasets are created, disjoint with respect to each other. Then, for each episode of the training phase, 100 beams are randomly selected from the train dataset. Then, during the test phase, 100 other beams are selected from the test dataset and the agent is evaluated on those beams. Simulations use 8 different environment in parallel, this way experience is shared throughout the training phase and statistics can be computed during evaluation.

As introduced in Table 8.3, two different policy optimization algorithms are considered: *Deep Q-Network* (DQN) [270] and *Proximal Policy Optimization* (PPO) [328]. The former is a value learning algorithm that has proven to be a good choice for discrete action space problems while the latter is a policy gradient algorithm [348] that has proven to achieve good results in a wide variety of real-world-based RL problems.

Similarly, two different policy networks are considered. In both cases, the network is first composed by two convolutional layers (first layer with 64 5x5 filters, second layer with 128 3x3 filters). Then, in one case the convolutional layers are followed by two fully-connected layers (first layer with 512 units, second layer with 256 units). In the other case the fully-connected layers are substituted by a 256-unit Long Short-Term Memory (LSTM) [170] network. In all cases, the network ends with an output layer that maps to the action space considered. Rectified linear units (ReLUs) activation layers and normalization layers are used in all cases.

Lastly, there are two versions of the training procedure being considered, with differ on the number of beams used during training with respect to the test beams. In one case, the number of training beams corresponds to the number of testing beams (100 in both cases for the baseline scenario). The second option doubles the number of training beams (200 for training, 100 for testing in the baseline scenario) in order

to make the task harder and potentially obtain a better policy. Additional beams are also sampled from the training dataset. The remaining implementation details are described in Appendix A.3.

### 8.5.2 Full enumeration analysis in the baseline scenario

The first analysis consists of carrying out a full enumeration analysis and test each possible combination of state-action-reward under the two possible training procedures for a total of 50,000 steps per training run. In addition, the full enumeration search is extended to include the *discount factor* $\gamma$ [348], a DRL hyperparameter. Three possible values are evaluated: 0.1, 0.5, and 0.9. A larger discount factor implies that the agent takes into account longer term effects of its actions, whereas lower discount factors are related to greedier policies. At this point the policy network and policy optimization algorithm are not considered, the CNN + MLP policy and DQN are used in each case, respectively. In total, 72 different designs are evaluated, and the result is shown in Table 8.4. This table presents the average number (across the 8 parallel environments) of successfully-assigned beams during test time (out of 100) for each combination of action, state, reward, discount factor $\gamma$, and number of training beams. For reference, these results are compared against a totally random policy, which achieves an average of 83.5 successfully-assigned beams.

One can observe that using the reward strategy *each* leads to better outcomes, and using the *grid* action space with the state space using *lookahead* does better on average. For the baseline case, there is no apparent advantage in using reward strategies that delay the reward until the end of the episode or rely on rollout policies; individual timesteps provide enough information to guide the policy optimization. To better understand the impact of the modeling decisions, below there are several significance tests that further analyze the trade-offs:

- There is no advantage in training with 200 beams instead of 100 ($P$-value = 0.08). Therefore, training with more beams does not make a better agent for this case. This is even less impactful when $\gamma = 0.1$ ($P = 0.90$), since the policy behaves greedily regardless of how many beams remain to be assigned. The strategy for placing the first 100 beams is similar both cases.
- The discount factor affects the performance of the policy ($P < 0.001$), but there is no significant performance difference when using $\gamma = 0.1$ or $\gamma = 0.5$ ($P = 0.83$). The performance worsens when $\gamma = 0.9$. The learned policy relies more

**Table 8.4:** Average number of successfully-assigned beams out of 100 from the test dataset in the baseline scenario for each combination of action, state, reward, discount factor $\gamma$, and number of training beams. A random policy achieves 83.5.

| | | **100-beam training** | | | | **200-beam training** | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **Grid** | | **Tetris-like** | | **Grid** | | **Tetris-like** | |
| **Reward** | $\gamma$ | **Lookahead** | **Without** | **Lookahead** | **Without** | **Lookahead** | **Without** | **Lookahead** | **Without** |
| **Each** | **0.1** | **99.8** | 95.6 | 92.0 | 95.8 | 98.9 | 97.9 | 87.5 | 98.2 |
| | **0.5** | 98.9 | 96.2 | 90.5 | 96.6 | 97.6 | 94.4 | 80.8 | 90.9 |
| | **0.9** | 96.6 | 93.1 | 81.5 | 66.9 | 95.2 | 93.4 | 70.2 | 66.6 |
| **Final** | **0.1** | 47.5 | 53.9 | 68.8 | 67.4 | 23.8 | 20.9 | 52.5 | 45.5 |
| | **0.5** | 46.6 | 56.0 | 39.9 | 70.1 | 21.1 | 21.2 | 67.0 | 69.9 |
| | **0.9** | 54.9 | 67.2 | 71.9 | 63.6 | 25.8 | 22.6 | 60.0 | 67.1 |
| **Rollout** | **0.1** | 19.5 | 17.6 | 45.2 | 45.2 | 2.5 | 2.9 | 50.9 | 34 |
| | **0.5** | 9.1 | 3.0 | 67.9 | 36.1 | 2.6 | 1.9 | 41.1 | 20.2 |
| | **0.9** | 2.2 | 2.6 | 37.8 | 47.5 | 2.2 | 3.2 | 34.6 | 24.1 |

on a greedy behavior to be successful, which is consistent with other methods that have addressed the problem [291].

- With $\gamma \leq 0.5$, the *each* reward strategy, and *grid* action space, the state space affects the performance ($P < 0.001$). It is better to include the lookahead layer.
- With $\gamma \leq 0.5$, the *each* reward strategy, and *tetris-like* action space, the state space affects the performance ($P < 0.001$). Not including the lookahead layer (*without* in the table) in the state achieves better results in this case.
- With $\gamma \leq 0.5$ and the *each* reward strategy, using the *grid* state space is a better alternative than using *tetris-like* ($P < 0.001$), since it supports more flexibility to make an assignment.
- Overall, the *tetris-like* action space appears to be less sensitive to the reward strategy chosen, especially when comparing across the simulations using the *rollout* reward strategy.

The lookahead layer plays an important role together with the *grid* action space, since the agent has total freedom to place a beam, and therefore being informed of the remaining assignments is beneficial. In contrast, in the case of the *tetris-like* action space, the agent is limited by the initial random placement of the beam, and therefore the information of what is to come is not as useful. This is an example of how different representations can interact with each other.

### 8.5.3 Scalability analysis

After assessing the DRL system works for the baseline scenario, the next step is focusing on the high-dimensionality challenge and analyze the impact scalability has on the agent. To that end, this section considers a scenario with 500 beams, 7 satellites, $N_P = 2$, $N_{FR} = 8$, $N_{BW} = 80$, $\gamma = 0.1$, and the *each* reward function, and compare all 4 combinations for the state and action representations. In this case each time the model is trained for a total of 200k timesteps and random set of 500 beams are sampled from both the train and test datasets. The results of this experiment are shown in Table 8.5, which are averaged across the 8 parallel environments and compared against a random policy, which achieves 429.9 successfully-assigned beams.

The main conclusion of this analysis is that when the dimensionality of the problem increases, the DRL system actually achieves a better outcome using the *tetris-like* action space, as opposed to the 100-beam case. The *grid* action space does even worse than random, which is due to the large amount of actions (1,280 for this case)

**Table 8.5:** Average number of successfully-assigned beams out of 500 in the test data using the *each* reward function and $\gamma = 0.1$. A random policy achieves 429.9.

| Action and state representation | Number of successfully-assigned beams |
|---|---|
| Grid and Lookahead | 320.1 |
| Grid and Without | 328.1 |
| Tetris-like and Lookahead | 461.6 |
| Tetris-like and Without | **478.9** |

and an inappropriate exploration-exploitation balance. Overall, there is no impact in terms of the state representation in this example ($P = 0.64$), although there is when conditioning on the *tetris-like* action space ($P < 0.001$); the agent does better *without* the lookahead layer. These results prove that relying on a single representation for a specific real-world problem might not be a robust strategy during operation. In the specific case of the frequency assignment problem, sticking to use cases with less than 100 beams as the majority of the literature would have rendered different conclusions. Understanding the limitations of specific representations is essential in order to make these models deployable.

### 8.5.4 Considering different policy networks and optimization algorithms

The last two DRL design elements considered in Table 8.3 that remain to be studied are the policy network and the policy optimization algorithm. In the real-world RL literature, typical design choices for these two components rely heavily on the progress by the domain-agnostic RL and the Deep Learning communities and rarely involve domain expertise in the design process. This is also the case for the choices considered in this experiment. The analyses of the previous section are now extended, the goal is to compare the baseline DQN algorithm and CNN + MLP against other alternatives.

As introduced at the beginning of the results section, in the case of the policy optimization algorithm, the Proximal Policy Optimization (PPO) algorithm [328] is chosen to be compared against DQN [270]. PPO is a policy gradient and on-policy algorithm and DQN is a value learning and off-policy algorithm. PPO optimizes the policy on an end-to-end fashion, by doing gradient ascent on its parameters according to a function of the cumulative reward over an episode. Additionally, it clips the gradients to avoid drastic changes to the policy. DQN focuses on optimizing the

prediction of the value of taking a certain action in a given state; it then uses these predictions to choose an action. Also, it stores all the agent's experience in a *replay buffer* and makes use of it over time by "replaying" past episodes and training on them. OpenAI's baselines [93] are used to implement each method.

Regarding the policy network, the second design option is to substitute the MLP block for a 256-unit LSTM. By comparing the CNN + MLP option against the CNN + LSTM option we can evaluate the impact of using a recurrent network as part of the policy. These structures take advantage of temporal dependencies in the data and therefore can potentially perform better in sequential problems.

Table 8.6 shows the results of using both policy networks and both policy optimization algorithms for 4 different cases: the 100-beam scenario using the *grid* action space with both state representations, the 500-beam scenario using the *tetris-like* action space with both state representations, and two additional 1,000-beam and 2,000-beam scenarios for the *tetris-like* action space *without* using the lookahead layer in the state representation. For each case the random policy successfully assigns 83.5, 429.9, 799.7, and 1,137.7 beams on average, respectively. Since the previous section has concluded the *tetris-like* action space better suits high-dimensional scenarios, only its performance is explored in the thousand-beam range. In all cases, the *each* reward strategy and $\gamma = 0.1$ is used. All simulations belonging to the same scenario are trained for an equal number of timesteps.

In the scenarios with 1,000 beams or less, one can observe there is no significant advantage in using PPO over DQN, although DQN consistently outperforms PPO in each scenario. Looking at the 2,000-beam scenario, however, the performance difference emerges. Note that the 1,000-beam and 2,000-beam simulations use the same values for $N_P$, $N_{FR}$, and $N_{BW}$. Therefore, in the 2,000-beam case, the performance worsening occurs during the assignment of the last 1,000 beams. During the last timesteps of such a high-dimensional scenario, having a prediction over multiple actions, as DQN does, proves to be a better approach, as opposed to PPO's method, which relies on the single action that the policy provides.

The same occurs if we compare the CNN+MLP policy (MLP in the table) against the CNN+LSTM policy (LSTM in the table). Given the greedy behavior of the policy, having the LSTM's hidden state does not offer any advantage when placing the last 1,000 beams in the 2,000-beam scenario. Although allowing more training iterations could help reducing the performance gap, these results prove that the choices of the policy and the policy optimization algorithm are especially important under certain circumstances. For the frequency assignment problem in the context of mega-

**Table 8.6:** Average number of successfully-assigned beams in the test data when comparing different combinations of policy networks and policy optimization algorithms for different use case dimensionalities. The *each* reward function and $\gamma = 0.1$ are used in all cases.

| Action and state representation | DQN MLP | PPO MLP | PPO LSTM |
|---|---|---|---|
| 100 beams, Random: 83.5 $N_P = 2$, $N_{FR} = 2$, $N_{BW} = 20$ | | | |
| Grid, Lookahead | **99.8** | 94.9 | 94.9 |
| Grid, Without | 95.6 | 97.5 | 91.5 |
| 500 beams, Random: 429.9 $N_P = 2$, $N_{FR} = 8$, $N_{BW} = 80$ | | | |
| Tetris-like, Lookahead | 461.6 | 460.5 | 470.0 |
| Tetris-like, Without | **478.9** | 478.0 | 465.1 |
| 1,000 beams, Random: 799.7 $N_P = 2$, $N_{FR} = 10$, $N_{BW} = 100$ | | | |
| Tetris-like, Without | **962.3** | 936.0 | 957.0 |
| 2,000 beams, Random: 1,137.7 $N_P = 2$, $N_{FR} = 10$, $N_{BW} = 100$ | | | |
| Tetris-like, Without | **1,746.0** | 1,139.0 | 967.0 |

constellations, operators care about the thousand-beam range. To help visualizing the performance of these models, Figure 8-8 shows the assignment results for one of the 8 environments in the 2,000-beam, DQN, and CNN+MLP scenario. A total of 1,766 beams are successfully assigned.

### 8.5.5 Non-stationarity analysis

Finally, this section addresses the second real-world RL challenge considered, non-stationarity, and analyzes its impact on trained DRL models. In the context of the frequency assignment problem, one interesting aspect of non-stationarity is the phenomena that might change between training to operation, such as the bandwidth distribution, the number of beams, or the constraint distribution. This section focuses on the bandwidth distribution for its analysis. Specifically, the goal is to study how the performance changes when the test environment includes beams with more average bandwidth demand than those in the training set. The analysis consists of carrying out simulations in which the average bandwidth demand per beam in the test set

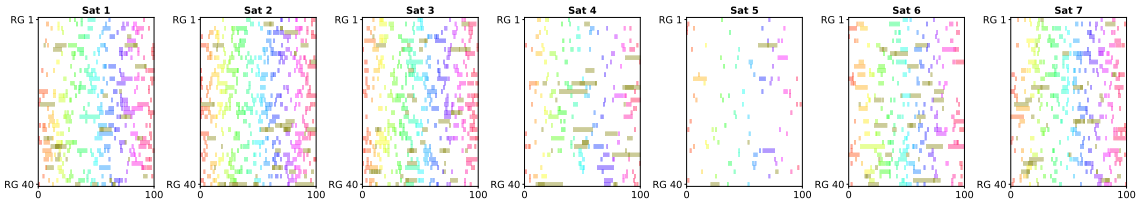



**Figure 8-8:** 2,000-beam Frequency Plan example obtained by the DQN-based agent using the CNN+MLP policy, the *tetris-like* action space, the state representation *without* the lookahead layer, the *each* reward function, and $\gamma = 0.1$. Golden cells indicate non-successful assignments (i.e., a constraint is violated), any other color indicates a successful assignment. 1,766 beams are successfully-assigned in this case. RG in the vertical axis corresponds to a combination of frequency reuse and polarization.

is 2 times and 4 times larger. This is done for the 100-beam scenario and the *grid* action space, as well as for the 500-beam scenario and the *tetris-like* action space. For both cases, both state representations are compared and the random policy is used to compare. The outcome of this analysis can be found in Table 8.7.

**Table 8.7:** Average number of successfully-assigned beams in the test data when comparing different imbalances between train and test data. The *each* reward function, the DQN algorithm, the CNN+MLP policy, and $\gamma = 0.1$ are used in all cases.

| **Action, State representation** | **Same demand** | **2-times demand** | **4-times demand** |
|---|---|---|---|
| 100 beams, $N_P = 2$, $N_{FR} = 2$, $N_{FS} = 20$ | | | |
| Grid, Lookahead | **99.8** | **84.9** | 52.6 |
| Grid, Without | 95.6 | 84.4 | 65.3 |
| Random | 83.5 | 81.3 | **71.9** |
| 500 beams, $N_P = 2$, $N_{FR} = 8$, $N_{FS} = 80$ | | | |
| Tetris-like, Lookahead | 461.6 | **453.0** | 249.0 |
| Tetris-like, Without | **478.9** | 434.0 | **337.0** |
| Random | 429.9 | 400.0 | 227.0 |

The results show that, as expected based on the literature on real-world RL challenges [100], non-stationarity does negatively affect the performance of trained DRL agents for the frequency assignment problem. In this case, it correlates with the average bandwidth demand per beam difference between the train and test data. In the 100-beam scenario, the performance gap between the random policy and the agent is reduced for the 2-times case, and then random performs better in the 4-times case.

Something similar occurs in the 500-beam scenario, although the random policy

does not beat the agent in any case. This is a consequence having a bigger search space and the more frequent use of the actions *up* and *down* in the *tetris-like* action space to make changes to the resource group assignment rather than changing the assignment within the same resource group. When analyzing the actions taken during test time for this latter scenario, the *up* and *down* actions were taken, on average, 2.5 times more than the *left* and *right* actions. Furthermore, non-stationarity affects the usefulness of the lookahead layer when used in combination with the *tetris-like* action space. In the analysis from Table 8.5, it is observed that not using it is a better alternative, whereas in the 2-times demand case the agent does better with it. There is a certain advantage in knowing what is to come in non-stationary scenarios, although there is a limit to how beneficial the lookahead layer is, as observed in the 4-times demand case.

## 8.6 Discussion of results

The proposed DRL system and its design variations have shown a quantitatively good outcome in the majority of scenarios considered. On average, the DRL agent has successfully assigned 99.8% of beams in the 100-beam case, 95.8% in the 500-beam case, 96.2% in the 1,000-beam case, and 87.3% in the 2,000-beam case. The main advantage of this approach is that evaluating a neural network is substantially faster than relying on other algorithms such as metaheuristics, while still showing a good performance. This has allowed the proposed system to scale up to the thousand-beam range, which many methods from the literature fail to do. To make up for the constraints that are not fulfilled, operators could use a subsequent "repairing" algorithm to address the conflicting beams, which would require much less time since there are less beams to allocate and most of the conflicts can be solved locally.

The interesting insight that comes from these numbers is that designing for robustness is not aligned with designing for performance, as the specific models for each scenario correspond to different design variations that use different state and action representations. In that sense, the goal of this chapter was not to treat design components as elements to be reported but as additional elements to be tuned. Given the importance of high-dimensionality in real-world satellite operations, evaluating robustness entailed focusing on cases with hundreds to thousands of beams, as opposed to works that limit themselves to cases with less than 100 beams. The conclusions on the importance of design variations for this problem could not have been extracted without explicitly designing for robustness in addition to performance. This

is aligned with some of the ideas discussed around real-world RL and the domain-specific research thrust. Evaluating the models under real-world conditions focused on robustness is at least as important as achieving or improving upon the state-of-the-art on performance-oriented benchmarks.

The results have also provided insights on the limitations resulting from environment non-stationarity. This chapter validates the RL domain-agnostic community's observations on the detrimental effect of non-stationarity on trained policies. It is uncommon to find these type of analyses on the domain-specific side. If the goal is to make DRL deployable, then it is essential that contingency cases caused by non-stationarity are properly addressed for the problems being considered. This can be done either by capturing sources of non-stationarity in the training data, or by devising strategies to mitigate its negative impact during real-time operations, which might entail relying on other methods at the same time.

Overall, the findings show that DRL is a potential method to address the frequency assignment problem in real operations, especially because of its speed in decision-making. However, no single model is able to outperform the rest in all scenarios, and the best approach for each of the six core design elements depends on the features of the operation environment. While DRL has the potential to solve future complex problems in the aerospace industry, it is also important to reflect on the necessity of designing appropriate models and training procedures for both performance and robustness, understanding the applicability of such models, and reporting the main trade-offs.

## 8.7 Chapter summary and Contributions

This chapter has explored the design of DRL systems for robustness from the perspective of a domain-specific problem: frequency assignment for satellite constellations. This addresses four of the research opportunities identified in Chapter 2: 1) focusing on applications beyond the realm of robotics, 2) automating parts of the design process, 3) prioritizing robustness in conjunction with performance in design, and 4) understanding the trade-offs between different design choices.

The initial part of the chapter has focused on presenting the context of the frequency assignment problem in satellite communications, highlighting the key motivation behind employing DRL for this issue —the demand for real-time or near real-time decision-making in high-dimensional contexts, a requirement traditional methodologies struggle to fulfill. This has been followed by a review of the related literature,

revealing that a considerable majority of studies, including those proposing DRL, fail to effectively encapsulate the flexibility and scalability of contemporary constellations or to accommodate rapid computing requirements.

The next section has outlined the problem statement, comprised of four different decision variables per beam and four types of constraints. A key outcome of this section has been the representation of the frequency assignment problem as a decision grid. Then, to gain mathematical intuition of this problem, I have proposed an Integer Linear Programming (ILP) formulation that, if combined with a commercial ILP solver, is capable of producing a complete frequency plan, given sufficient computing time.

Then, the DRL system to solve the frequency assignment problem has been presented. This system has been designed around six different core design decisions that have a major impact in the outcome of a training run: the policy network, the policy optimization algorithms, the state representation, the action space, the reward function, and the training configuration. Different alternatives for each of these components have been proposed with the objective of creating a search space of diverse designs to be studied. Then, grid search has served as a means of automating the search for the best choices.

**Table 8.8:** Summary of the best result achieved by the proposed DRL system (as a % of total number of beams successfully assigned) for each combination of degree of dimensionality considered and demand distribution at test time. The case *same demand* corresponds to scenarios in which non-stationarity is not present, otherwise non-stationarity entails larger average demand at test time.

| **Test demand** | **100 beams** | **500 beams** | **1,000 beams** | **2,000 beams** |
|---|---|---|---|---|
| Same demand | 99.8% | 95.8% | 96.2% | 87.3% |
| 2x demand | 84.9% | 90.6% | - | - |
| 4x demand | 71.9% | 67.4% | - | - |

Section 8.5 has presented the analytical evaluation of this chapter, which has focused around four sets of experiments. Table 8.8 shows the summary of the best result obtained for each of the experimental configurations considered. First, I have carried out a full enumeration analysis using grid search on the decisions for the state representation, the action space, the reward function, and the training configuration. In addition, I have included three values for the discount factor; 72 different designs have been evaluated, with 8 random seeds each. This first experiment has identified

a reward function, a training configuration, and a value for the discount factor to downselect.

The next experiment has focused on increasing the dimensionality of the problem by 5 and determining whether the relative results among design variations hold. While the *grid* action space has proved to be better suited in low-dimensionality scenarios, this experiment has demonstrated that the *tetris-like* action space is more robust against scalability. Then, I have kept analyzing the latter action space in scenarios in the thousand-beam range, also including the variations for the policy network and the policy optimization algorithm.

Finally, the last set of experiments has focused on the challenge of non-stationarity, examining what happens during operations if the DRL agent is deployed but the demand distributions changes. By considering cases with two and four times increased demand, I have shown that adding a *lookahead* layer in the state representation is more robust. The implications of the complete analysis and the process followed for this real-world RL problem have been discussed in Section 8.6.

The specific contributions of this chapter are the following:

**Contribution 8.1** Proposed a novel Integer Linear Programming (ILP) formulation that describes the modern frequency assignment problem for multibeam satellite constellations. This formulation can be directly integrated with a commercial ILP solver.

**Contribution 8.2** Developed a DRL system to address the frequency assignment problem in satellite communications that optimizes decision-making for both performance and robustness.

**Contribution 8.3** Identified a design variation of the DRL system that trains agents that achieve high performance in low-dimensional scenarios.

**Contribution 8.4** Identified an alternative design variation of the DRL system that trains agents able to demonstrate robustness against scalability, up to the thousand-beam range.

**Contribution 8.5** Conducted an in-depth analysis of the performance versus robustness trade-off among different design variations, with specific regard to the challenges posed by high-dimensionality and non-stationarity.

# Chapter 9

# Case Study 2: Deep Reinforcement Learning for Molecular Optimization

This chapter covers the second real-world use case for which DRL system design is concretely analyzed: molecular optimization. Similarly to the case of frequency assignment, the objective of this chapter is to better understand how the design of DRL systems for this applications influences the performance and robustness of agents by directly considering domain-specific performance and robustness benchmarks. To that end, Section **9.1** introduces again the problem and discusses the related literature. Then, Section **9.2** formulates the problem, highlighting the role of generative models in the process of discovering new compounds. Section **9.3** addresses the benchmarking process, covering the experimental setup, the choice of models to compare, and the results, which are later discussed in Section **9.4**. Finally, Section **9.5** concludes the chapter and presents the main contributions.

## 9.1 Introduction

The second real-word use case for which this dissertation analyzes aspects of robustness is DRL for molecular optimization, also known as drug design or *de novo* drug design. As introduced in Chapter 7, given the enormous size of the molecular space, finding new compounds that optimize a set of properties is prohibitively expensive when only relying on the chemists' experience and intuition. Computational approaches to generate new molecules have been long studied, from metaheuristic algorithms such as evolution [95, 384] to modern deep learning methods [97, 104, 195, 196, 239]. Still, property targeting remains a considerable challenge for state-of-the-art methods, es-

pecially when looking for multiple properties at the same time. To close this gap, DRL has been proposed as a method to produce undiscovered molecules with better properties, since the reward function serves as a way of biasing the generative process. However, while recent results are promising, so far there is still only one reported use and success of DRL in end-to-end pharmaceutical trials [412].

The literature on DRL for molecular optimization reveals a scenario similar to that of the frequency assignment problem: experimental setups explore basic property targeting use cases, design choices are seldom discussed, and there is no clear consensus on which design choices are superior. In addition, since many of the tests occur under non-disclosure agreements, it is not clear how different methods in the literature rank in terms of robustness. This problem also presents several real-world challenges in combination: high-dimensionality, generalizability, multi-objectiveness, representation, and long trajectories. While there have been efforts in the literature to create thorough benchmarking protocols [45,421] and apply them [131], their use is still not widespread and designs do not seem to converge. For example, some studies treat molecules as 2D or 3D graphs, while other authors propose methods that treat them as tokenized sequences such as SMILES. These differences apply to many other components of the DRL system such as the action space, the reward function, or the policy optimization algorithm.

This chapter examines trade-offs among the principal design choices of methods in the literature and benchmarks some of the most popular models in order to gain design insights. Specifically, it compares different text-based and graph-based models in more than 20 tests from the GuacaMol benchmarking suite [45] across six different types of benchmark, including rediscovery tests, similarity tests, isomer discovery tests, median molecule tests, multi-property optimization tests, and structural tests.

### 9.1.1 Related Work

The related work section examines recent studies on the application of DRL to drug design. This section limits to DRL and the design of molecules composed by atoms and bonds. Works that use other methods for property targeting and the application of DRL in other biological domains (e.g., proteins) are left out of its scope. Table 9.1 summarizes the literature on DRL. Given the goal is to better understand design choices for this problem, the table breaks down each work by how molecules are represented, the definition of the state, the action space, the reward functions, the policy optimization algorithms, and the experimental setups that were considered.

**Table 9.1:** Summary of relevant works in the literature studying DRL for drug design / molecular optimization. logP = partition coefficient of a solute between octanol and water (indicator of solubility). p-logP = logP - SA. SA = Synthetic accessibility. QED = Quantitative estimate of druglikeness. PSA = Polar surface area. MW = Molecular weight. MC = Monte Carlo. †Independent experiments separated by commas.

| Reference | Molecular repr. | State repr. | Action space | Reward function† | Algorithm | Experimental setup |
|---|---|---|---|---|---|---|
| Stahl et al. [341] | Fragments | Binary encoding of fragments | Fragment to change | Binary, if property in range | Vanilla actor-critic | Optimize for logP, PSA, and MW |
| You et al. [397] | Graph | Current graph + candidate scaffolds | New edge or new scaffold to be added | Direct property and binary reward if property in range | PPO | Optimize for logP, QED, or MW |
| Tan et al. [353] | SMILES + fragments | SMILES strings of two fragments | 4 Full token sequence generation | Direct property and binary reward | REINFORCE + regularization | Inhibit JAK3, QED + SA, logP |
| Zhou et al. [418] | Morgan fingerprint | Current graph converted to Morgan fingerprint | Atom addition, bond addition, bond removal | Direct property | Double DQN | Optimize for p-logP, QED |
| Guimaraes et al. [146] | SMILES | Current sequence | Next token | Weighted combination of property and discriminator reward (uses MC sampling in the process) | REINFORCE | Optimize for QED, SA, logP |
| Olivecrona et al. [281] | SMILES | Current sequence | Next token | Direct property scaled in [-1, 1] | REINFORCE + regularization | Avoid sulphur, celecoxib generation, Activity against DRD2 |

Continued on next page

**Table 9.1 Continued from previous page**

| Reference | Molecular repr. | State repr. | Action space | Reward function† | Algorithm | Experimental setup |
|---|---|---|---|---|---|---|
| Popova et al. [306] | SMILES | Current sequence | Next token | Direct property | REINFORCE | Optimize for melting temp., logP, JAK2 inhibition, num. benzene rings |
| Zhavoronkov et al. [412] | SMILES | Current sequence + latent representation | New token | Bio-activity prediction using self-organizing maps | REINFORCE | Find DDR1 kinase inhibitors (real-world), optimize for logP, QED |
| Horwood et al. [174] | Morgan fingerprints | Complete molecule after $k$ reactions, $k \geq 0$ | Chemical reactions (hierarchical) | Direct property | A3C | Optimize for DRD2 activity, p-logP, QED |
| Atance et al. [19] | Graph | Current graph | Add new node or connect last node to other node | Direct property or binary reward | REINFORCE + regularization | Optimize for DRD2 activity, QED, number of atoms |
| Gottipati et al. [145] | Morgan fingerprints + MACCS public keys | Complete molecule after $k$ reactions, $k \geq 0$ | Reactant | Direct property | TD3 + k-NN [99] | Activity against HIV (3 different scores), optimize for QED, logP, |
| Flam-Shepherd et al. [119] | 3D graphs | Current graph + set of fragments | Next fragment | Differential energy | PPO | Generate molecules from various fragment sets |

The table shows that there exists great diversity of choices for each of the different components considered. For example, some studies treat molecules as graphs, following their natural structure; others treat them as tokenized sequences like SMILES [383]; and others also consider using fingerprints [51] of the molecules. The state representation generally follows from the molecular representation considered, therefore it consists of partial graphs [397], partial sequences [306], and fingerprints from partial molecules [418]; but it can also be encoded using binary trees [341], or MACCS keys [145]. The action spaces can be divided into spaces that append some substructure to the molecule like atoms or scaffolds [397], spaces that select the next token in the sequence [281], spaces that only add edges [119], and spaces that select reactants [145] and thus find new molecules via chemical reactions.

Then, one of the design components with the largest variety is the reward function. While in many cases it depends on the specific experiment carried out, some authors define the reward as a direct measurement of the property [418] using popular cheminformatics tools like RDKit [225], others define a binary reward [341], and in other works an activity model is first learned via supervised learning and then is used as reward function [174, 412]. The DRL algorithms that have been proposed in the literature range from simple REINFORCE implementations [412] and variations [353], to more complex versions of actor-critic algorithms. PPO is one of the preferred choices [397], and in some cases versions of DQN are considered [418]. As pointed out in Chapter 1, domain-specific research rarely proposes new algorithmic methods; this is an example of such trend. Finally, the experimental setups are mostly single-objective, in few cases multiple objectives are considered via parametrized functions [353]. Most of the papers consider basic molecular properties such as indicators of solubility (logP), synthetic accessibility scores (SA), and quantitative estimates of druglikeness (QED). Fewer works address specifically showing activity against a biotarget [145, 281, 353]. Finally, the work in [412] is the only one that reports results on the complete drug generation pipeline, from design to synthesis.

Although the columns in Table 9.1 already demonstrate the differences among the different methods in the literature, there are other design factors that papers treat differently:

- There is no widespread criteria on when to stop the generation process of a molecule, some works fix the number of steps [418], others have a termination state [341], others add a stop signal in the action space [397], or an *end-of-sequence* (EOS) character [281, 306].

- Some authors use the original versions of algorithms like REINFORCE or PPO, others add regularizers [353] or change the loss function [397, 418].
- There are works that consider an initial supervised learning stage in which a prior model is learned [146, 281, 397, 412]. The sizes of the datasets used to train these models range from 250k to 1.5M molecules.
- In some cases the validity of the molecules during the generation process is enforced by adding constraints to the action space [418], in other cases validity is a penalty added to the reward function [281].
- Some authors allow states to represent incomplete molecules [306, 397], others only consider state representations for which molecules are complete [145, 353].
- When to provide rewards to the agent is also a design choice, some authors only provide nonzero rewards when reaching the terminal state [174, 281], others guide the agent in the middle of the generation process [397].

The literature review demonstrates that there is a lot of diversity in how the DRL systems are designed for the molecular optimization problem. Given the lack of real-world-based benchmarking, there is little knowledge of which design choices lead to more robust agents that can operate across more tasks. The rest of the chapter addresses this issue for a selection of models and design choices.

## 9.2 Problem formulation

The problem of molecular optimization entails the exploration and discovery of novel molecules with desired properties. Molecules can be viewed as complex graphs where nodes represent atoms and edges represent bonds. Each node and edge can have a type, reflecting different kinds of atoms and bonds respectively. The problem involves investigating large, high-dimensional search spaces, given that a molecule's properties are determined by its atomic configuration. This section introduces a general mathematical formulation of the problem to provide a rigorous understanding and to simplify its inherent complexity.

**The search space of molecules** Let's define the search space of molecules as $\Omega$, which is the set of all possible molecules defined by all possible atomic configurations. In the context of the graph perspective, each molecule $\omega \in \Omega$ can be represented as a graph $G = (V, E)$ where $V$ is the set of nodes or atoms and $E$ is the set of edges or bonds. Each node $v \in V$ has an associated type $t(v)$, representing the atomic element

of the node, with the set of all possible atomic elements denoted as $A$. Similarly, each edge $e \in E$ has an associated type $t(e)$, representing the type of the bond (e.g., simple, double), with the set of all possible bond types denoted as $B$. Hence, $t(v) : V \to A$ and $t(e) : E \to B$.

Each molecule is associated with a property of interest $Y(\omega)$, which is a mapping from $\Omega$ to a real number, i.e., $Y : \Omega \to \mathbb{R}$. The goal is to find the molecule $\omega^* \in \Omega$ for which $Y(\omega)$ achieves an optimum (maximum or minimum), formally expressed as:

$$\omega^* = \arg\max_{\omega \in \Omega} Y(\omega) \quad \text{for maximization, or}$$
$$\omega^* = \arg\min_{\omega \in \Omega} Y(\omega) \quad \text{for minimization.}$$

However, directly optimizing $Y(\omega)$ is challenging due to the high-dimensionality of the search space, the combinatorial nature of $\Omega$, and the cost of evaluating $Y(\omega)$, which requires expensive experiments. Instead, a generative model $G$ is used as an intermediary step to guide the search.

**The generative model** The generative model is defined as $G$ and is parameterized by $\theta$, it learns the underlying distribution of $\Omega$, i.e., $P_G(\omega; \theta)$. In practice, the model searches over a space $Z$, which is a mapping from a lower-dimensional latent space to $\Omega$, i.e., $G : Z \to \Omega$. To learn this latent space, it relies on a dataset with an empirical distribution $D$ and the learning process consists of minimizing a divergence measure $D(P_G||P_D)$. Once the model is learned, the problem can be transformed into a search in the latent space $Z$. However, $P_G(\omega; \theta)$ may not directly align with the optimization objective $Y(\omega)$. Therefore, the real challenge lies in adapting the learned generative model to guide the search towards regions in $\Omega$ that optimize the property $Y(\omega)$.

The transformed problem thus involves finding $z^* \in Z$ such that $\omega = G(z^*)$ optimizes $Y(\omega)$, formally expressed as:

$$z^* = \arg\max_{z \in Z} P(G(z; \theta)) \quad \text{for maximization, or}$$
$$z^* = \arg\min_{z \in Z} P(G(z; \theta)) \quad \text{for minimization.}$$

This reformulation mitigates the problem's complexity, as the latent space $Z$ is typically much smaller than $\Omega$ and the generative model $G$ provides a structured way of exploring $\Omega$.

**Deep Reinforcement Learning** Now this considers the case in which the generative model $G$ is finetuned by means of a DRL system. The neural network of the model acts like the policy network $\pi$, which interacts with the environment represented by the latent space $Z$. The finetuning process begins with a state $s$, which is a point in the latent space $Z$. Depending on the implementation and the chosen representation, a point in the latent space might not always correspond to a molecule in $\Omega$, as some implementations build on partial molecules.

The agent then takes an action $a$ according to a policy $\pi(a|s;\theta)$, which is typically a stochastic function of $s$ and $\theta$. The action $a$ transforms the current state to a new state $s'$ in the latent space. The quality of the new molecule is evaluated by the reward function $r(s,a,s') = Y(G(s';\theta))$. The goal of the DRL model is to learn the policy that maximizes the expected cumulative reward:

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\pi(a|s;\theta)} \left[ \sum_{t=0}^{T} \gamma^t r(s_t, a_t, s_{t+1}) \right] \tag{9.1}$$

where $T$ is the time horizon, $\gamma \in [0,1]$ is the discount factor, and the expectation is over trajectories of states and actions generated by the policy $\pi$. By optimizing this objective, the DRL model can finetune the generative model to produce molecules that have higher values of the property of interest $Y(\omega)$. This framework provides a structured approach to explore the space of molecules and adapt the generative model based on the feedback from the environment.

It is worth noting that not all DRL approaches incorporate a finetuning process as outlined above. An alternative paradigm involves concurrent learning of the structure of the molecular space and the specifics of the targeted property. These methods aim to simultaneously capture the inherent chemical composition rules and the property optimization objective within a unified framework. While this approach avoids relying on a dataset $D$, it often poses other challenges due to the complex interplay between the generative process and the property optimization.

## 9.3 Benchmarking experiments

This section presents benchmarking experiments using the GuacaMol benchmarking suite [45] to assess the relative performance of different classes of DRL models for molecular optimization. To that end, it tries to capture some of the different design choices introduced in Table 9.1 by considering examples of text-based models, graph-

based models, and descriptor space models. The distribution of the section is the following: Section 9.3.1 presents the experimental setup; then Sections 9.3.2, 9.3.3, and 9.3.4 introduce each of the three model classes and the selected representative examples. Lastly, Section 9.3.5 presents the results of the benchmarking process.

### 9.3.1 Experimental setup

The experiments carried out in this section focus on two clear objectives. On one hand, they seek to compare different DRL designs for the molecular optimization problem when solving tasks that go beyond targeting individual simple properties such as the molecular weight. On the other hand, the experiments consist of a task set diverse enough to capture aspects of robustness of each of the designs being compared.

In order to provide a structured framework for the experiments, the choice of molecular representation is regarded as the critical design factor for comparing different models. Three different representations are considered: text-based, graph-based, and descriptor space-based. For each class, one or two methods are benchmarked on a broad range of tasks. The models are introduced in the sections that follow. Most of the implementation details are kept identical to the original papers, the exact configurations are reported in Appendix A.4.

Regarding the benchmark, instead of developing one from scratch, the GuacaMol benchmark [45] is used in the process. The GuacaMol benchmarking suite is a comprehensive framework developed by BenevolentAI for the evaluation of models in *de novo* drug design. It consits of a set of well-defined and meaningful tasks that mirror real-world drug discovery challenges. While it is gaining traction in the community as a way to compare different drug design methods [131], its use is still limited.

Specifically, GuacaMol consists of two main components: a benchmark suite and task-specific scoring functions. The benchmark suite comprises a collection of 20 tasks divided into six task types including, but not limited to, rediscovery of known drugs, lead optimization, scaffold hopping, and designing molecules with multiple concrete properties. The scoring functions are used to quantify the model's performance on each task, ensuring a fair and objective comparison between different models. These are directly used as reward functions for all the different methods considered. Additional details on the benchmarking suite are presented in the original paper [45].

In the following sections each of the three representation classes are presented and their representative examples introduced. In total, four different methods are

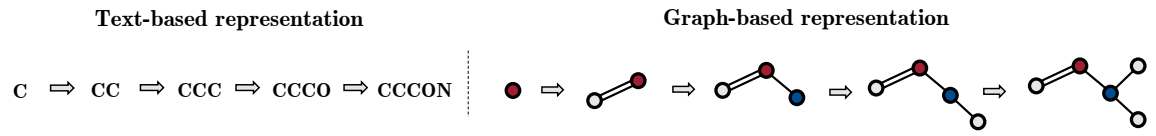



**Figure 9-1:** Text-based (left) and graph-based (right) representations of a molecule.

considered: ReLeaSE, REINVENT, GraphINVENT, and CDDD+PPO. In all of these models, a prior model is built via supervised learning. The GuacaMol benchmark also addresses this aspect, as it provides a dataset of molecules from the ChEMBL database [264] which all models use to learn the prior.

### 9.3.2 Text-based generative models

Text-based generative models leverage textual representations of molecules (Figure 9-1 left), such as SMILES strings [383], to learn and generate novel molecular structures. These methods capitalize on the extensive developments in natural language processing (NLP) and apply similar techniques to model the sequence of characters that represent a molecule.

Typically, DRL systems using on text-based representations rely on neural networks with recurrent structure, such as RNNs, LSTMs, gated recurrent units (GRUs), or, recently, transformer-based architectures. These models are usually trained to predict the next character in a SMILES string given the preceding characters, thereby learning the underlying syntactical rules of molecular structures.

Despite their success, text-based models have limitations. Not all generated strings correspond to valid molecular structures due to the specific syntax rules of SMILES, and the models may require additional steps to ensure validity of the generated molecules. Furthermore, they miss out on important structural information; two molecules with different molecular structure might be encoded by the same sequence. In addition, during the generation process, incomplete sequences might be useless and that usually leads designers to set intermediate rewards to zero.

Two text-based models will be compared: **ReLeaSE** [306] and **REINVENT** [281]. Both are based on SMILES strings, make use of recurrent architectures that generate the molecule one token at a time, and are first trained via supervised learning to build a prior model. ReLeaSE uses a stack-RNN and a GRU module and only provides nonzero rewards at the end of the generative process. The policy is optimized via REINFORCE [348], which accounts for the magnitude of the reward and the likelihood of the trajectory to make changes using the policy gradient method.

REINVENT only uses a MultiGRU module and optimizes the policy using a single update. To that end, the discount factor $\gamma$ is set to zero, which allows to sample multiple episodes at once. The algorithm uses experience replay of the best molecules found during the learning process and regularization techniques to 1) encourage exploration, 2) penalize deviations from the prior model. In their papers, ReLeaSE and REINVENT are used for optimizing melting temperature, solubility, and activity against JAK2; and celecoxib generation and activity against DRD2, respectively.

### 9.3.3 Graph-based generative models

In contrast to text-based models, graph-based generative models (Figure 9-1 right) directly treat molecules as graphs, where nodes represent atoms and edges represent bonds. These models seek to capture the graph-structured data of molecular structures, and thereby more accurately reflect the inherent structural properties of molecules.

Graph-based models generally leverage graph neural networks (GNNs) to encode the features of nodes and edges, capturing the local and global structure of the molecule. Then, they generate new molecules by adding nodes and edges to an existing graph, ensuring that each intermediate step represents a valid molecular structure. This way, intermediate structures can be exploited in the form of rewards.

Graph-based models inherently enforce the validity of generated structures, overcoming a key limitation of text-based models. However, they often face challenges in terms of computational complexity due to the need to manipulate graph-structured data.

One recent graph-based model is chosen to be benchmarked: **GraphINVENT** [19, 265]. The application of GraphINVENT for property targeting via DRL is also separated in a first supervised learning phase and a subsequent DRL phase. GraphINVENT's architecture consists of a gated-graph neural network (GGNN) [416] which first transforms initial feature vectors by means of graph operators and then outputs an action probability distribution (APD) by applying fully-connected layers on top of concatenated representations from the previous stage. The APD allows to add one atom to the molecule, connect the previously added atom to another atom, or terminate the process. The DRL stage is similar to REINVENT's; it is based on REINFORCE with the addition of regularizers that penalize deviations from the prior unless the reward is large. The experimental setup considered for GraphINVENT includes optimizing for QED, number of atoms, and activity against DRD2.

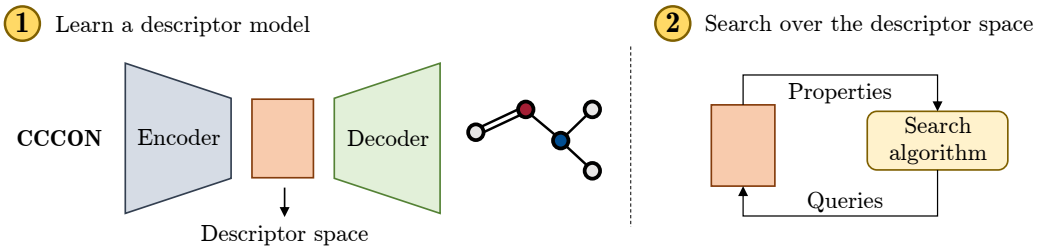



**Figure 9-2:** Descriptor space-based models for molecular optimization.

### 9.3.4 Descriptor space-based models

Another class of models for molecular optimization operates directly in descriptor spaces rather than the original molecular structure space. These models represent molecules using low-level, typically continuous descriptors, which capture relevant information about the molecules and their properties (see Figure 9-2).

The primary goal is to learn a transformation from the high-dimensional, discrete molecular structure space to a lower-dimensional, continuous descriptor space using neural networks. This is typically accomplished by training models to translate between different representations of the molecular structures (e.g., from SMILES to graph and vice versa) using encoders and decoders, thereby embedding the meaningful information into a compact, low-dimensional descriptor.

Once trained, we can generate new molecules by navigating in the continuous descriptor space and mapping points in this space back to the original molecular space. This approach allows for efficient exploration and optimization as the descriptor space is often smoother and better suited for a wide variety of search methods such as bayesian optimization [217] or particle swarm optimization [158].

However, one of the challenges with these models is ensuring the validity of the generated molecular structures. While the transformation from descriptor space to molecular space is generally smooth, it is not guaranteed to produce valid molecules at all points in the descriptor space, and additional steps may be required to enforce the validity of the generated molecules.

This dissertation takes a different approach for these methods and implements a DRL model to search over the descriptor space. The implementation of the descriptor space is based on the CDDD model [384] and PPO [328] without any modification (OpenAI baselines version [93]) is used to navigate the descriptor space. This method is labeled as **CDDD+PPO**.

### 9.3.5 Benchmark

Table 9.2 presents the result of each of the 4 methods compared (ReLeaSE, REINVENT, GraphINVENT, and CDDD+PPO) for all 20 tests of the GuacaMol benchmarking suite, which are divided into six types: rediscovery (3 tests), where the goal is to learn to generate a specific molecule that is not in the original ChEMBL database; similarity (3 tests), where the goal is to discover molecules that are structurally close to molecules chosen for each task; isomer discovery (2 tests), where the goal is to generate molecules that correspond to a specific target formula; median molecules (2 tests), which consists of conflicting tasks of maximizing similarity to two molecules; multi property optimization (8 tests), where the goal is to discover molecules that optimize for two to four different objectives at the same time; and structural changes (2 tests), which specifies certain structural patterns that generated molecules must fulfill. The table also includes the performance of four other models and the best performance registered in the ChEMBL database, all as reported in [45]. The other models in the table are: An implementation of PPO using a LSTM (LSTM PPO), a genetic algorithm operating over SMILES (SMILES GA), a genetic algorithm operating over graphs (Graph GA), and a Monte Carlo tree search algorithm operating over graphs (Graph MCTS). The reader is referred to the original publication [45] for a full description of these additional models.

An initial glance at the results reveals that no single method consistently outperforms the others in every task, underscoring the complexity and diversity of this real-world problem as represented by the GuacaMol benchmarks. Different methods display strengths and weaknesses based on the individual characteristics of each task, which agrees with what other studies relying on this benchmark conclude [45, 131].

Focusing on DRL, REINVENT emerges as the top performer in the majority of the tasks. However, its average performance across benchmarks is not the largest; the Graph GA model as reported in the original GuacaMol paper achieves a slightly better result (0.89 vs. 0.90, respectively).

Interestingly, while graph-based models are often perceived as more sophisticated and superior, we can observe this is not always the case. For example, REINVENT, which utilizes SMILES strings, outperforms GraphINVENT, a graph-based model that uses the same learning algorithm. This observation coincides with other non-DRL studies [131] and underscores that success in molecular design might not simply be a matter of representation type.

The novel CDDD+PPO model, though not the top scorer, consistently outper-

**Table 9.2:** Performance of each DRL model considered on the 20 tests of the GuacaMol benchmark. Best in each row is **bolded**. LSTM = Long short-term memory. PPO = Proximal policy optimization. GA = Genetic algorithm. MCTS = Monte carlo tree search. Absent scores correspond to experiments that failed during runtime due to memory issues. † Performance values as reported in [45].

| **Benchmark type** | **Benchmark** | **Best of dataset**† | **LSTM PPO**† | **SMILES GA**† | **Graph GA**† | **Graph MCTS**† | **ReLeaSE** | **REINVENT** | **CDDD PPO** | **GraphINVENT** |
|---|---|---|---|---|---|---|---|---|---|---|
| Rediscovery | Celecoxib | 0.51 | **1.00** | 0.73 | **1.00** | 0.36 | 0.46 | **1.00** | **1.00** | 0.49 |
| | Troglitazone | 0.42 | **1.00** | 0.52 | **1.00** | 0.31 | 0.42 | **1.00** | 0.70 | 0.30 |
| | Thiothixene | 0.46 | **1.00** | 0.60 | **1.00** | 0.31 | 0.48 | **1.00** | 0.67 | 0.44 |
| Similarity | Aripiprazole | 0.60 | **1.00** | 0.83 | **1.00** | 0.38 | 0.70 | **1.00** | **1.00** | 0.47 |
| | Albuterol | 0.72 | **1.00** | 0.91 | **1.00** | 0.75 | 0.86 | **1.00** | **1.00** | 0.61 |
| | Mestranol | 0.63 | **1.00** | 0.79 | **1.00** | 0.40 | 0.52 | **1.00** | **1.00** | 0.42 |
| Isomer discovery | $C_{11}H_{24}$ | 0.68 | 0.99 | 0.83 | 0.97 | 0.41 | - | **1.00** | 0.99 | 0.13 |
| | $C_9H_{10}N_2O_2PF_2Cl$ | 0.75 | 0.88 | 0.89 | 0.98 | 0.63 | - | **1.00** | **1.00** | 0.46 |
| Median molecules | Camphor, menthol | 0.33 | 0.44 | 0.33 | 0.41 | 0.23 | 0.35 | **0.45** | 0.39 | 0.24 |
| | Tadalafil, sildenafil | 0.35 | 0.42 | 0.38 | **0.43** | 0.17 | 0.23 | **0.43** | 0.35 | 0.18 |
| Multi property optimization | Osimertinib | 0.84 | 0.91 | 0.89 | 0.95 | 0.78 | - | **0.96** | 0.91 | 0.77 |
| | Fexofenadine | 0.82 | 0.96 | 0.93 | **1.00** | 0.70 | - | **1.00** | 0.99 | 0.70 |
| | Ranolazine | 0.79 | 0.86 | 0.88 | 0.92 | 0.62 | - | **0.94** | 0.89 | 0.73 |
| | Perindopril | 0.58 | **0.81** | 0.66 | 0.79 | 0.39 | 0.46 | 0.79 | 0.72 | 0.45 |
| | Amlodipine | 0.70 | 0.89 | 0.72 | 0.89 | 0.53 | 0.61 | **0.91** | 0.83 | 0.51 |
| | Sitagliptin | 0.51 | 0.55 | 0.69 | **0.89** | 0.46 | 0.41 | 0.74 | 0.78 | 0.29 |
| | Zalepon | 0.55 | 0.67 | 0.41 | **0.75** | 0.49 | 0.47 | 0.68 | 0.62 | 0.42 |
| | Valsartan | 0.26 | 0.98 | 0.55 | **0.99** | 0.04 | - | **0.99** | - | 0.00 |
| Structural changes | Scaffold hop | 0.74 | 0.99 | 0.97 | **1.00** | 0.59 | - | **1.00** | 0.77 | 0.48 |
| | Deco hop | 0.93 | 0.99 | 0.89 | **1.00** | 0.48 | - | **1.00** | 0.96 | 0.70 |
| | **Average** | 0.61 | 0.87 | 0.72 | **0.90** | 0.45 | 0.50 | 0.89 | 0.82 | 0.44 |

forms the best molecule in the dataset, demonstrating its capability to generate improved solutions. This indicates that the combination of continuous drug-descriptor latent spaces and DRL algorithms like PPO can be a promising avenue for future research.

## 9.4 Discussion of results

The results validate that DRL is well-suited to address the task of molecular optimization, although its robustness remains challenged, as many of the methods presented in the literature are tailored to specific use cases. While it is still not clear if generalization with respect to the problem can be achieved with a single method, the results show that REINVENT is able to score the top performance in the majority of tests; it achieves a perfect score in rediscovery tasks, similarity tasks, isomer discovery tasks, and structural change optimization tasks. While representing molecules as SMILES strings might entail loss of structural information, the syntactical rules of this representation can be exploited to successfully complete many simple tasks.

Further analysis reveals REINVENT's particular weakness in some of the multi-property optimization tasks. This exposes a lack of robustness against real-world RL challenges in drug discovery, which is often characterized by multi-objectiveness and out-of-distribution optimization. As the complexity of the task increases and involves optimizing for multiple objectives at the same time —something common in real-world settings— REINVENT loses the top spot to the Graph GA model reported in [45].

However, graph-based models that use DRL might still be too brittle for this specific problem. While the results when considering a GA show that the graph structure provides substantial advantages over SMILES, GraphINVENT displays deficiencies in these tasks. While it showed promising results in the original paper —which the model used in this dissertation was able to reproduce— design for graph-based models might be too tailored for specific use cases, which in the case of GraphINVENT consisted of single-objective optimizing for molecular weight and QED. However, it is worth mentioning that exhaustive hyperparameter tuning was not performed for any of these models, which could potentially affect the reported performances.

While these findings provide valuable insights, it is important to bear in mind that the GuacaMol benchmark is based on scoring functions derived from chemoinformatics libraries such as RDKit. These results, thus, reflect the models' performance in this specific benchmark context, but do not necessarily imply similar performance in

real-world drug discovery pipelines. The application and testing of these models in practical scenarios, with real-world constraints and requirements, remain open areas for further exploration and publication.

Overall, the outcomes of this chapter reveal that robustness is not at the center of current design practices for DRL systems for molecular optimization, some of the model designs being highlighted in the literature might actually fall short when evaluated across a range of broader tasks. With use cases characterized by high-dimensionality and multi-objectiveness as central challenges in drug discovery, evaluating robustness necessitates focusing on complex tasks rather than settling for simpler ones.

## 9.5 Chapter summary and Contributions

This chapter has delved into the design of DRL systems for molecular optimization, the second real-world problem considered in this dissertation. Despite its complexity exceeding that of the frequency assignment in satellite communications, it serves as a valuable case study for exploring DRL design beyond the realms of robotics and control applications, and understanding the degree to which current design practices overlook robustness in favor of performance. These aspects represent two of the research opportunities pinpointed in Chapter 2.

The first part of the chapter has introduced the problem, articulating its significance and the role DRL can play in improving current drug design pipelines. It has also addressed the alignment, or lack thereof, of current proposed methods with real-world deployability, indicating that models in existing literature are rarely subjected to a broad array of tests reflecting the problem's diversity, and design choices are not frequently discussed. These two bottlenecks have been further studied in the related work section, revealing that among the dozen published DRL models for this task, over five different molecular representations, state representations, action spaces, reward functions, and algorithms exist. Furthermore, the lack of consensus regarding experimental setups is even larger, as those are seldom identical and typically explore only simplistic single-objective tasks, such as optimization for logP or QED.

The next part of the chapter has introduced a mathematical formulation for the problem, structuring it around the molecular search space, the generative model, and its finetuning through DRL. Then, Section 9.3 has shifted the focus to benchmarking experiments, initially introducing the experimental setup founded on the GuacaMol benchmarking suite. This suite provides 20 distinct benchmarks split into six cate-

gories, a set of scoring functions, and a database facilitating the construction of prior models via supervised learning. Next, four different models compared in this section have been presented: two text-based models, ReLeaSE and REINVENT; one graph-based model, GraphINVENT; and a novel method, a descriptor space-based model that employs PPO to navigate the descriptor space generated by CDDD [384].

Then, the benchmarking results have been presented, which have been later discussed in Section 9.4. The findings have underscored that no single method dominates the rest; while REINVENT achieves the highest score in most tests, it does not obtain the best overall score. The multi-property optimization benchmarks, which arguably mirror real-world use cases closest, remain challenging for all DRL models. Graph-based models like GraphINVENT, despite impressive results in their respective papers, may be excessively tailored to specific scenarios. The novel CDDD + PPO implementation achieves good results, despite not being the top scorer. Although these results and methods warrant further analysis and benchmarking, this dissertation emphasizes that the design for robustness is often not prioritized.

The specific contributions of this chapter are the following:

**Contribution 9.1** Proposed a novel method for molecular optimization which involves navigating a descriptor space using Proximal Policy Optimization. This approach achieves an average score of 0.82 in the GuacaMol benchmarking suite.

**Contribution 9.2** Performed comprehensive benchmarking of four different DRL models for molecular optimization using the GuacaMol benchmarking suite, which offers a thorough evaluation across 20 varied drug discovery tasks.

**Contribution 9.3** Assessed the performance versus robustness trade-off in the context of molecular optimization, with a focus on different molecular representations reported in the literature. The analysis revealed that challenges associated with multi-objectiveness and out-of-distribution optimization are still not adequately addressed, especially within complex tasks.

# Chapter 10

# Conclusion

Reinforcement Learning (RL) is increasingly gaining attention in numerous real-world domains as a tool to automate and enhance many decision-making processes. However, this escalating interest is not mirrored by a corresponding increase in the reported deployment of RL systems in real-world environments. Currently, significant impacts are only observed when large amounts of computational and human expert resources are available, creating a substantial bottleneck for its widespread study and deployment. Consequently, a pivotal question within the broader RL community is how to make RL technology more accessible and deployable without incurring considerable costs.

A key factor in the slow progress of real-world RL is the need to address several challenges that complicate the learning process. While the research community is primarily focused on achieving new performance benchmarks, these challenges necessitate a robustness perspective that aligns more closely with the deployment requirements of practitioners. This dissertation has tackled the question of how we can enhance the robustness of RL systems to real-world problems and phenomena.

To take steps towards answering this complex question, three overarching objectives have been established at the beginning of this dissertation. Firstly, as robustness is not a comprehensively-studied issue in RL, one goal has been to characterize different aspects of robustness that impede the deployment of RL systems in the real world. Secondly, another objective has been to develop new RL design methods that prioritize robustness and complement existing work focused on performance. Lastly, the third goal has involved applying the conceptual frameworks and design principles explored in this dissertation to address specific real-world problems.

This chapter summarizes the key findings and contributions that have emerged from addressing these objectives. It begins with an overview of the dissertation's

specific objectives and a summary of each chapter (Section **10.1**). This is followed by a detailed recap and discussion of the Thesis contributions (Section **10.2**). The chapter concludes by outlining the areas of future work that this dissertation has opened up (Section **10.3**).

## 10.1 Thesis summary

**Chapter 1** has begun by presenting the context and historical evolution of RL, highlighting Deep RL as a pivotal moment in RL's history. Then, the lack of deployments and the need for increasing the robustness of RL systems have been underscored as the primary motivating factors for this dissertation. This dissertation complements two existing research thrusts: domain-agnostic and domain-specific, both of which have been introduced in the first chapter.

In this context, **Chapter 2** has comprehensively reviewed the literature on real-world RL, focusing on three key topics: the domain-agnostic challenges of real-world RL and strategies to mitigate them, the application of AutoML frameworks to RL, and reported successes of RL in the real world. This review has led to the identification of seven research opportunities that have been explored throughout the remainder of the dissertation. These include extending the robustness research scope to use cases beyond the realm of robotics (Chapters 8 and 9), addressing combined challenges of real-world RL (Ch. 5 and 6), investigating the automation of design for robustness (Ch. 4, 5, 6, and 8), prioritizing robustness in design to avoid problem tailoring (Ch. 7, 8, and 9), exploring the problem of non-convergence of agent design in many applications (Ch. 4), studying the trade-offs between performance and robustness (Ch. 6 and 8), and developing a comprehensive framework that covers all aspects of real-world RL (Ch. 3).

**Chapter 3** has presented a pioneering DRL roadmap, aiming to provide a comprehensive view of all elements that impact the interaction between a DRL system and a real-world environment. This has been driven by the need to address the often overlooked aspects related to the implementation and operation of DRL systems in research studies. The chapter has not provided a recipe for perfecting DRL in real-world applications, but rather has identified all the components involved in the process. The roadmap links a real-world task with the system-level goals that need to be achieved, and is divided into three phases: design, implementation, and operation. Each phase has been thoroughly characterized and discussed. The design phase explores ideas on design automation versus human-driven design, environment-related

design decisions, agent-related decisions, and training configurations. The implementation phase focuses on the concepts of robustness, system-level generalization, and operability, and introduces strategies to bridge the deployment gap in RL research. The operation phase, often neglected in literature, addresses critical aspects such as the selection of agents for deployment, the potential sim-to-real gap, and the necessity for continual learning.

**Chapter 4** has delved into one of the elements of the roadmap, the state space, and has examined current design practices in real-world RL, aiming to enhance them through mutual information (MI) analyses. The chapter has focused on feature-based state spaces; it has highlighted the excess of features and the lack of consensus on feature sets as present issues in the literature. Looking at feature selection strategies, MI has been identified as a popular method for feature selection in ML and, to a certain extent, in RL. However, two important shortcomings in the use of MI in RL are its infrequent utilization in domain-specific RL studies and the failure of domain-agnostic studies to consider scenarios where the policy changes as the agent learns. The work in this chapter has consisted of deriving a mathematical bound on how much the MI can change between two different policies for the same Markov Decision Process. This bound has been also applied for the specific case of a policy changing until convergence and the removal of features from the state space. The chapter has concluded with a real-world use case, the Traffic Signal Control problem, validating the observations around MI and emphasizing the need to observe the MI along the complete learning curve, as it is an inexpensive way of selecting features in large spaces.

**Chapter 5** has addressed two significant research opportunities in real-world RL: the focus on combined real-world RL challenges and the automation of the RL design process. To this end, the chapter has introduced MetaPG, an AutoML method that automates the design of new actor-critic loss functions represented as computational graphs to optimize multiple objectives. To guide the search, MetaPG relies on several components including evolution mechanisms with several strategies to reduce cost, the multi-objective optimization algorithm NSGA-II, a search space encoding a diverse set of operations, and multiple fitness scores. These fitness scores can encode independent metrics of robustness, leading to the discovery of Pareto Fronts of new algorithms that maximize and trade all objectives considered.

**Chapter 6** has evaluated MetaPG on a specific use case on optimizing for single-task performance, zero-shot generalizability, and stability across independent training runs. Overall, this chapter has proved that MetaPG can evolve algorithms that im-

prove upon performance, generalizability, and stability in different practical settings. In the process, it uses two fitness scores explicitly encoding performance and generalizability, and implicitly encouraging stability. Relying on different environments from the RWRL Environment suite, OpenAI Gym, and the Brax physics simulator, MetaPG has found new RL algorithms that improve upon a popular baseline, SAC, in all objectives considered for different environments. On average, MetaPG has evolved Pareto Fronts that achieve a 4.2%, a 13.4%, and a 67% increase in performance, generalizability, and stability, respectively. In addition, some of the evolved algorithms are capable of transferring to unseen environments, achieving at least similar results compared to SAC, better in some cases. Furthermore, by examining the equations of the evolved loss functions, interpretations of the performance and generalizability biases have been derived, proving that MetaPG provides valuable information about the search outcome.

**Chapter 7** has set the groundwork for a detailed exploration of domain-specific aspects of DRL robustness and its co-existence with performance. The chapter has presented the typical workflow of domain-specific RL research when developing new DRL solutions for a specific application, identifying three key gaps that could explain why robustness often gets sidelined: a lack of full understanding of the real-world RL challenges present in the problem, failure to define the subproblems that matter most for robustness, and inadequate testing of the agent across different subproblems to better understand the performance versus robustness trade-offs. This chapter has also introduced the two real-world use cases that have been studied in this dissertation: frequency assignment for satellite constellations and molecular optimization.

**Chapter 8** has delved into the design of DRL systems for robustness from the perspective of a domain-specific problem: frequency assignment for satellite constellations. This is an important use case for DRL beyond the field of robotics, as traditional methods struggle to fulfill the requirements of modern constellations, namely real-time or near real-time decision-making in high-dimensional contexts. However, in the literature, DRL has neveer been tested for robustness against high-dimensionality and non-stationarity, two key real-world challenges of this problem. The chapter has presented a new Integer Linear Programming formulation for the problem, and has proposed a DRL system to solve it. By conducting a grid search over six major design decisions of the DRL system, a design variation has been found to solve the problem for low-dimensional use cases, achieving a 99.8% success rate in a 100-beam case. However, this variation has not been able to scale, proving that different design choices were required in order to prioritize robustness against high-dimensionality. A design

variation that has primarily relied on a different action space has achieved a 87.3% success rate in a 2,000-beam case. Similarly, when specifically testing for robustness against non-stationarity due to demand shifts, a different design variation has been proved to achieve better results. Overall, this chapter has demonstrated that, while robustness is crucial for deployment, domain-specific research seldom prioritizes it when designing new DRL methods.

Finally, **Chapter 9** has focused on the design of DRL systems for molecular optimization, another impactful real-world problem in which DRL can substantially contribute to accelerate drug development pipelines. The chapter has addressed the alignment, or lack thereof, of current proposed methods with real-world deployability, as many experimental setups in the literature are narrow and mostly concern single-property optimization. The main outcome of this chapter has been a thorough benchmarking process of four different DRL models using the GuacaMol benchmarking suite. The results have underscored that no single method dominates the rest, and the multi-property optimization benchmarks, which arguably mirror real-world use cases closest, remain challenging for all compared models. One of the methods compared is novel and integrates a descriptor space from the literature [384] with a PPO-based search, achieving an average score of 0.82 on the GuacaMol benchmark.

## 10.2 Contributions

This dissertation has made several contributions that extend the current understanding of RL robustness for real-world problems. These are summarized in Table 10.1, which enumerates them according to the chapter each is related to. The contributions are divided into four types: *conceptual*, which correspond to new concepts and frameworks that have been proposed throughout the dissertation; *theoretical*, which correspond to new theory-based proofs and lemmas derived in this dissertation; *methodological*, which correspond to new methods, algorithms, and implementations that have come out of this dissertation's work; and *practical*, which corresponds to advances and insights for specific domains that have constituted the use cases of this dissertation.

**Table 10.1:** Summary of Thesis contributions. *Contribution X.Y* corresponds to contribution number Y in chapter X.

| Contribution | Description | Type |
|---|---|---|
| Contribution 2.1 | Identified and characterized two RL research thrusts, namely domain-agnostic and domain-specific, that currently contribute to the real-world applicability of RL. | Conceptual |
| Contribution 2.2 | Conducted a comprehensive review of real-world RL literature, which revealed several areas of research that are currently unaddressed by any of the existing research thrusts. | Conceptual |
| Contribution 3.1 | Proposed a DRL roadmap that identifies and breaks down key elements that influence the process of using DRL in real-world applications. | Conceptual |
| Contribution 3.2 | Applied the DRL roadmap to characterize the relationship between the deployment of DRL systems in the real-world and their robustness, system-level generalization, and operability. | Conceptual |
| Contribution 4.1 | Characterized the limitations of current state space design practices for real-world RL problems, identifying an excess of features and a lack of consensus as two important shortcomings. | Conceptual |
| Contribution 4.2 | Identified a lack of study of feature selection via mutual information in RL in the context of a changing policy and motivated its consideration by proving the policy can substantially influence the observed mutual information. | Conceptual |
| Contribution 4.3 | Derived a mathematical bound for how much the mutual information between state features and rewards can change when considering two different policies acting on the same Markov Decision Process. | Theoretical |
| Contribution 4.4 | Derived a mathematical bound for how much the mutual information between state features and rewards can change as the policy converges in the context of an RL algorithm. | Theoretical |
| Contribution 4.5 | Outlined the possible updates on mutual information bounds when removing a set of features from the state space. | Theoretical |
| Contribution 4.6 | Provided a better feature set for two use cases of the RESCO benchmark for Traffic Signal Control. Demonstrated this feature set is also identifiable via mutual information. | Practical |

Continued on next page

**Table 10.1 Continued from previous page**

| Contribution | Description | Type |
|---|---|---|
| Contribution 4.7 | Validated the observations of mutual information changes during policy learning using a real-world application and motivated the need to compute the mutual information at different points of the convergence process to get reliable estimations of feature relevance. | Practical |
| Contribution 5.1 | Proposed MetaPG, a method that combines multi-objective evolution and a search language representing actor-critic RL algorithms as graphs to discover new loss functions that optimize a set of different RL objectives. | Methodological |
| Contribution 5.2 | Developed MetaPG's search space, which offers the flexibility to represent a wide range of actor-critic RL algorithms. | Methodological |
| Contribution 6.1 | Formulated MetaPG fitness scores to explicitly optimize for performance and generalizability and implicitly encourage stability. | Methodological |
| Contribution 6.2 | By running MetaPG, identified set of Pareto-optimal loss functions that have been evolved in different environments and outperform Soft Actor-Critic for such environments in terms of performance, generalizability, and stability. | Methodological |
| Contribution 6.3 | Provided, for the specific case of RWRL Cartpole, a comprehensive dataset of algorithms obtained throughout the complete evolution process (not only the Pareto Front). This dataset may be further analyzed to gain additional insights on algorithmic changes and trade-offs. | Methodological |
| Contribution 7.1 | Discussed the typical workflow domain-specific RL research follows when proposing new real-world RL methods, as well as the robustness-related gaps that are frequently overlooked. | Conceptual |
| Contribution 8.1 | Proposed a novel Integer Linear Programming (ILP) formulation that describes the modern frequency assignment problem for multibeam satellite constellations. This formulation can be directly integrated with a commercial ILP solver. | Methodological |
| Contribution 8.2 | Developed a DRL system to address the frequency assignment problem in satellite communications that optimizes decision-making for both performance and robustness. | Methodological |
| Contribution 8.3 | Identified a design variation of the DRL system that trains agents that achieve high performance in low-dimensional scenarios. | Practical |

Continued on next page

**Table 10.1 Continued from previous page**

| Contribution | Description | Type |
|---|---|---|
| Contribution 8.4 | Identified an alternative design variation of the DRL system that trains agents able to demonstrate robustness against scalability, up to the thousand-beam range. | Practical |
| Contribution 8.5 | Conducted an in-depth analysis of the performance versus robustness trade-off among different design variations, with specific regard to the challenges posed by high-dimensionality and non-stationarity. | Practical |
| Contribution 9.1 | Proposed a novel method for molecular optimization which involves navigating a descriptor space using Proximal Policy Optimization. This approach achieves an average score of 0.82 in the GuacaMol benchmarking suite. | Methodological |
| Contribution 9.2 | Performed comprehensive benchmarking of four different DRL models for molecular optimization using the GuacaMol benchmarking suite, which offers a thorough evaluation across 20 varied drug discovery tasks. | Practical |
| Contribution 9.3 | Assessed the performance versus robustness trade-off in the context of molecular optimization, with a focus on different molecular representations reported in the literature. The analysis revealed that challenges associated with multi-objectiveness and out-of-distribution optimization are still not adequately addressed, especially within complex tasks. | Practical |

These contributions address the different research opportunities on real-world RL that domain-agnostic and domain-specific research thrusts do not currently consider. These research opportunities were identified in Chapter 2 and the mapping to the contributions outlined in each chapter of this dissertation is shown in Table 10.2.

## 10.3 Future work

Several directions of future work have been identified during the completion of this dissertation. They can be grouped into the following areas:

**Improving MetaPG's efficiency** MetaPG, in its current state, is a composite system with multiple complex components. The version presented in this dissertation lays a foundation, demonstrating that this methodology can automate the design of multiple RL objectives, with an emphasis on robustness. As highlighted in Chapter

**Table 10.2:** Mapping of the thesis chapter contributions to the research opportunities identified in Chapter 2.

| Research opportunity | Ch. 3 | Ch. 4 | Ch. 5 | Ch. 6 | Ch. 7 | Ch. 8 | Ch. 9 |
|---|---|---|---|---|---|---|---|
| There is little research on designing for real-world robustness outside of robotics use cases. | | | | | | ✓ | ✓ |
| There is little research on combined aspects of robustness for real-world RL. | | | ✓ | ✓ | | | |
| Designing for robustness is currently human-driven, which is costly when systems scale. | | ✓ | ✓ | ✓ | | ✓ | |
| Research prioritizes performance over robustness, this comes as a result of excessive problem tailoring. | | | | | ✓ | ✓ | ✓ |
| The designs of RL agents for specific applications do not converge. | | ✓ | | | | | |
| There is little understanding of design choices and trade-offs. | | | | ✓ | | ✓ | |
| Research in RL for real-world problems does not capture the full picture of designing, implementing, and operating RL systems. The design aspect is prioritized over implementation and operation, as research generally only reports on results. | ✓ | | | | | | |

6, each component within MetaPG could potentially be refined to further optimize its overall performance. Presently, despite the inclusion of consistency checks, the hashing process, and the environment hurdles, MetaPG's operation remains resource-intensive. A promising direction for future research could involve devising more cost-effective methods for exploring the loss function search space, without compromising on loss function expressiveness. Additionally, cost efficiency could also be augmented from the standpoint of cross-environment performance. Throughout this dissertation, MetaPG has been meta-trained in isolation with single environment classes, which has led to marginal improvements in transfers across diverse environments. To foster

increased algorithmic reusability, an optimized, cost-reduced MetaPG could be meta-trained simultaneously with a wider diversity of environments.

**Expanding the use cases for MetaPG** Future research avenues for MetaPG could involve the execution of a broader spectrum of use cases. As discussed in Chapter 5, various fitness scores could be evaluated for MetaPG, including but not limited to sample-efficiency and non-stationarity. Furthermore, integrating more than two fitness scores into MetaPG is a possibility, albeit one that would increase the cost of the search process. Lastly, given MetaPG's demonstrated ability to discover new performance thresholds, another potential research direction could involve deploying it for complex problems, particularly those where identifying high-performing methods presents a significant challenge.

**Automating the design of other RL components for robustness** MetaPG has successfully unified the automation of RL loss function design and RL robustness into a single method. Additional elements of the algorithm such as the gradient descent method could be also incorporated into the search space. Pivoting from searching over the loss function space, future research could explore the integration of existing AutoRL algorithms for other components, such as the reward function or the training curriculum, with robustness optimization. The automation of RL and the specification of system-level goals tied to robustness could lead to impactful developments in RL.

**Characterizing the operation space for a problem** Figure 3-7, set within the context of the DRL roadmap, has introduced the concept of operability and has emphasized the necessity for a DRL system to adapt to all subproblems considered essential by a practitioner or operator for effective deployment. While the ensuing ideas derived from this concept have been articulated at a conceptual level too, an interesting avenue of future research could involve mapping out the set of subproblems for a specific real-world problem. This characterization could then serve as a deployment benchmark, offering a valuable reference point for emerging research advances.

**Optimizing the state space for other use cases** Chapter 4 has relied on the Traffic Signal Control (TSC) problem as a practical example to investigate feature selection through mutual information (MI). Although the TSC scenario served as an

effective means to evaluate the theoretical implications derived in this dissertation, broadening the application spectrum to encompass other use cases would be desirable, particularly those entailing larger feature sets.

**Tailoring mutual information bounds to specific RL algorithms** The mathematical bounds derived in Chapter 4 have not assumed any particular underlying learning algorithm. However, in a practical context, there exists a diverse set of RL algorithms and algorithm classes, some of which impose restrictions on the frequency and magnitude of policy changes (e.g., trust region policy optimization algorithms [328]). Incorporating assumptions about the underlying RL algorithm into the derivation process could yield updated, specific mathematical bounds for each case. This approach would offer a more customized understanding of feature selection for cases that depend on a particular RL algorithm.

**Evaluating the DRL system for frequency assignment on additional constellations** Although the evaluation approach adopted in Chapter 8 encompassed an unprecedented degree of scalability, gauged by the number of beams in the constellation, the experimental framework only encapsulated a small portion of the flexibility inherent in modern constellation systems. A more comprehensive understanding of the performance and robustness of DRL within this real-world problem could be garnered by expanding the range of scenarios and use cases examined. This expansion should not only incorporate additional values for the number of beams but also include a variety of constellations. Similarly, this dissertation has solely addressed one facet of non-stationarity in this problem, namely, demand shifts. Exploring the modeling of other operational phenomena, such as changes in the user base or sudden system failures, could yield further interesting insights.

**Expanding the design search space for real-world use cases** Lastly, a potential research direction is to explore considerably larger design spaces than those evaluated in the two real-world use cases presented in this dissertation. In each instance, a handful of choices have been considered for each design decision, a process which has effectively highlighted that current practices may not align well with robustness. However, in practice, this represents a modest degree of flexibility. The work with MetaPG has demonstrated that search spaces, when unconstrained by human design biases, can be extensive and potentially yield substantial improvements in both performance and robustness.

# Appendix A

# Experimental setups

This appendix presents details on the experimental setups and configurations used in the different experimental work of this dissertation. Section A.1 presents the setup for Chapter 4, Section A.2 concerns Chapter 6, and Sections A.3 and A.4 for the real-world use cases in Chapters 8 and 9, respectively.

## A.1 State space analyses

This section focuses on the experimental configurations related to the research on state space analyses, presented in Chapter 4. It first covers the configurations used for the PyBullet experiments and then the setup for the Traffic Signal Control ones.

### A.1.1 PyBullet experiment configurations

Table A.1 shows the configuration used for each of the experiments with the PyBullet environments [75]. Then, specific details about the environments' feature spaces are shown in Table A.2.

### A.1.2 Traffic Signal Control experiment configurations

Table A.3 shows the configuration used for each of the experiments with the RESCO benchmark [20] for the Traffic Signal Control problem. Then, specific details about the environments' configuration are shown in Table A.4.

**Table A.1:** Configuration used for the PyBullet experiments.

| Parameter | Value |
|---|---|
| Discount factor $\gamma$ | 0.99 |
| Maximum episode length | 1,000 |
| Algorithm | PPO [328] |
| Epochs | 200 |
| Steps per epoch | 4,000 |
| Clip ratio | 0.2 |
| Learning rate policy $\pi$ | 0.0003 |
| Learning rate value function $v$ | 0.001 |
| Training iterations $\pi$ | 80 |
| Training iterations $v$ | 80 |
| $\lambda$ | 0.97 |
| Target KL | 0.1 |
| Policy $\pi$ | MLP, $64 \times 64$ |
| Value function $v$ | MLP, $64 \times 64$ |
| Activation | Tanh |

**Table A.2:** Feature space details for PyBullet environments.

| Parameter | Number of features |
|---|---|
| XYZ body position (all) | 3 |
| XYZ body velocity (all) | 3 |
| Roll (all) | 1 |
| Pitch (all) | 1 |
| Joint positions Hopper | 3 |
| Joint velocities Hopper | 3 |
| Contact points Hopper | 1 |
| Joint positions Ant | 8 |
| Joint velocities Ant | 8 |
| Contact points Ant | 4 |
| Joint positions Humanoid | 17 |
| Joint velocities Humanoid | 17 |
| Contact points Humanoid | 2 |

**Table A.3:** Configuration used for the Traffic Signal Control experiments.

| Parameter | Value |
|---|---|
| Discount factor $\gamma$ | 0.99 |
| Algorithm | DQN |
| Batch size | 32 |
| Number of episodes | 100 |
| Target network update frequency | 500 |
| Policy | 2×2 conv. layer + MLP 64 × 64 |

**Table A.4:** Environment configuration in the RESCO benchmark.

| Parameter | Number of features |
|---|---|
| Step length | 10 seconds |
| Length yellow signal | 3 seconds |
| Simulation length | 1 hour |

## A.2 MetaPG

This section presents the configuration elements for MetaPG's use case studied in Chapter 6, which consisted of optimizing for performance, zero-shot generalizability, and stability. First, the environment details are introduced, followed by the training configurations and hyperparameter tuning procedures.

### A.2.1 MetaPG environment configurations

Chapter 6 uses multiple environments for the experiments: Cartpole and Walker from the RWRL Environment Suite [100], Pendulum from OpenAI Gym [43], and Ant and Humanoid from the Brax physics simulator [126]. Table A.5 lists the training configuration used for each and the parameters that are used to assess the generalizability of the policies. The parameters that are not listed are fixed to the default values for the environment in question.

In the case of Pendulum and the Brax environments, there is more than one perturbation parameter; the generalizability score is computed by first sweeping through the perturbation values for one, taking the average $f_{gen_1}$, repeating the same process for the others to compute $f_{gen_2}$, ..., $f_{gen_P}$, and then computing the average of both to

get the final score, i.e., $f_{gen} = (f_{gen_1} + ... + f_{gen_P})/P$, where $P$ is the total number of different parameters that are perturbed.

For the parameters that undergo perturbations, the specific value used in the training configuration is selected based on the following criteria: in the case of the RWRL Environment Suite, the training value is taken from [100], which corresponds to the default value as described in each benchmark; in the case of Pendulum, the default values given by the environment are chosen for the training configuration; in the case of Brax, the rationale is to select training configurations that represent a baseline scenario (i.e., no mass variations, normal friction, and normal torque).

### A.2.2 MetaPG training configurations

An individual from the population encoding an RL algorithm in the form of a graph is evaluated by training an agent using such algorithm. The implementation used in that process is based on an ACME agent [171] for the RWRL and Gym environments, and an implementation based on the Brax physics simulator [126] for the Brax environments. The configuration of the training setup is shown in Table A.6 for RWRL and Gym environments, and in Table A.7 for the Brax environments.

### A.2.3 Hyperparameter tuning in MetaPG - RWRL and OpenAI Gym environments

In the experiments with the RWRL Environment suite and OpenAI Gym Pendulum, once an evolution experiment is over and the evolved algorithms are meta-validated, they are compared against: 1) ACME SAC [171], and 2) other RL algorithms that have been evolved in a different environment. To that end, for each ACME benchmark and evolved algorithm transfer, algorithm hyperparameters are tuned. Since there are two fitness scores (performance and generalizability adjusted for stability), two hyperparameter configurations are selected: those that lead to the best stability-adjusted performance and best stability-adjusted generalizability scores, respectively. These two configurations are denoted as the best performer and best generalizer, respectively. To that end, a grid search is done across the sets of hyperparameters listed in Table A.8. This process is only carried out once the evolution is over; the warm-start algorithm is not hyperparameter-tuned before evolution.

**Table A.5:** Environment parameters and perturbations used for MetaPG experiments.

| Environment parameter | Value |
|---|---|
| **RWRL Cartpole** | |
| Rollout length | 1,000 |
| Min. return | 0 |
| Max. return | 1,000 |
| Training episodes | 150 |
| Perturbation parameter (PP) | Pole length |
| PP Default value | 1.0 |
| PP Generalizability values | 0.1 to 3.0 in steps of 0.1 |
| **RWRL Walker** | |
| Rollout length | 1,000 |
| Min. return | 0 |
| Max. return | 1,000 |
| Training episodes | 225 |
| Perturbation parameter (PP) | Thigh length |
| PP Default value | 0.225 |
| PP Generalizability values | .1, .125, .15, .175, .2, .225, .25, .3, .35, .4, .45, .5, .55, .6, .7 |
| **Gym Pendulum** | |
| Rollout length | 2,000 |
| Min. return | -2,000 |
| Max. return | 0 |
| Training episodes | 100 |
| Perturbation parameter 1 (PP1) | Pendulum mass |
| PP1 Default value | 1.0 |
| PP1 Generalizability values | .1, .2, .4, .5, .75, 1.0, 1.5, 2.0, 3.0, 5.0, 7.5, 10.0 |
| Perturbation parameter 2 (PP2) | Pendulum length |
| PP2 Default value | 1.0 |
| PP2 Generalizability values | .1, .2, .4, .5, .75, 1.0, 1.5, 2.0, 3.0, 5.0, 7.5, 10.0 |
| **Brax environments** | |
| Rollout length | 1,000 |
| Min. return | 0 |
| Max. return | 10,000 (Ant) and 14,000 (Humanoid) |
| Training episodes | 1,000 |
| Perturbation parameter 1 (PP1) | Mass coefficient |
| PP1 Default value | 1.0 |
| PP1 Generalizability values | 0.8 to 1.2 in steps of 0.05 |
| Perturbation parameter 2 (PP2) | Friction coefficient |
| PP2 Default value | 1.0 |
| PP2 Generalizability values | 0.3 to 1.0 in steps of 0.05 |
| Perturbation parameter 3 (PP3) | Torque multiplier |
| PP3 Default value | 1.0 |
| PP3 Generalizability values | 0.5 to 1.0 in steps of 0.05 |
| Perturbation parameter 4 (PP4) | Combined parameters PP1, PP2, and PP3 |
| PP4 Default value | 1.0 for each |
| PP4 Generalizability values | Grid search over individual generalizability values |

**Table A.6:** RL Training setup for the RWRL and OpenAI Gym environments.

| Parameter | Value |
|---|---|
| Discount factor $\gamma$ | 0.99 |
| Batch size | 64 (RWRL Cartpole and Gym Pendulum) |
| | 128 (RWRL Walker) |
| Learning rate | $3 \cdot 10^{-4}$ |
| Target smoothing coeff. $\tau$ | 0.005 |
| Replay buffer size | 1,000,000 |
| Min. num. samples in the buffer | 10,000 |
| Gradient updates per learning step | 1 |
| n step | 1 |
| Reward scale | 5.0 |
| Actor network | MLP (256, 256) |
| Actor activation function | ReLU |
| Tanh on output of actor network | Yes |
| Critic networks | MLP (256, 256) |
| Critic activation function | ReLU |

### A.2.4 Hyperparameter tuning in MetaPG - Brax environments

In the experiments with Brax Ant and Brax Humanoid, given they are more costly environments to run, not all algorithms in the population are meta-validated. Instead, the best algorithms during meta-training are chosen and directly meta-tested with additional hyperparameter tuning. To that end, a grid search is done across the hyperparameters listed in Table A.9 and the configuration that maximizes the score of interest is selected for each case, as described in the previous section.

## A.3 Frequency assignment

Table A.10 presents the relevant hyperparameters for the frequency assignment experiments described in Chapter 8 and the values used in this dissertation.

**Table A.7:** RL Training setup for the Brax environments.

| Parameter | Value |
|---|---|
| Discount factor $\gamma$ | 0.95 |
| Batch size | 128 |
| Learning rate | $6 \cdot 10^{-4}$ |
| Target smoothing coeff. $\tau$ | 0.005 |
| Replay buffer size | 1,000,000 |
| Min. num. samples in the buffer | 1,000 |
| Gradient updates per learning step | 64 |
| Reward scale | 10.0 |
| Number of parallel environments | 128 |
| Actor network | MLP (256, 256) |
| Actor activation function | ReLU |
| Tanh on output of actor network | Yes |
| Critic networks | MLP (256, 256) |
| Critic activation function | ReLU |

**Table A.8:** Hyperparameter values considered during the tuning process for RWRL and OpenAI Gym environments.

| Hyperparameter | Values |
|---|---|
| Discount factor $\gamma$ | 0.9, 0.99, 0.999 |
| Batch size | 32, 64, 128 |
| Learning rate | $1 \cdot 10^{-4}$, $3 \cdot 10^{-4}$, $1 \cdot 10^{-3}$ |
| Target smoothing coeff. $\tau$ | 0.005, 0.01, 0.05 |
| Reward scale | 0.1, 1.0, 5.0, 10.0 |

## A.4 Molecular optimization

Finally, Table A.11 presents the relevant hyperparameters for the molecular optimization experiments described in Chapter 9 and the values used in this dissertation. Additional hyperparameter configurations can be found in the original papers for ReLeaSE [306], REINVENT [281], CDDD [384], and GraphINVENT [19, 265].

**Table A.9:** Hyperparameter values considered during the tuning process for Brax environments.

| Hyperparameter | Values |
|---|---|
| Discount factor $\gamma$ | 0.95, 0.99, 0.999 |
| Batch size | 128, 256, 512 |
| Learning rate | $1 \cdot 10^{-4}$, $6 \cdot 10^{-4}$, $1 \cdot 10^{-3}$ |
| Gradient updates per learning step | 32, 64, 128 |
| Reward scale | 0.1, 1.0, 10.0, 100.0 |

**Table A.10:** Hyperparameter values used in the frequency assignment experiments.

| Hyperparameter | Value |
|---|---|
| DQN batch size | 16 |
| DQN replay buffer size | 100x Num. beams |
| DQN target network update freq. | 100 |
| DQN initial exploration | 0.5 |
| DQN final exploration | 0.2 |
| PPO num. steps | 64 |
| PPO num. opt. epochs | 4 |
| PPO num. minibatches | 8 |
| PPO $\lambda$ | 0.8 |
| PPO clip factor | 0.2 |
| CNN network | 3 layers (64 5x5, 128 4x4, and 256 3x3 filters) |
| MLP network | 3 layers (512, 256, 128 units) |
| Activation | ReLU |
| Learning rate | 0.0001 |

**Table A.11:** Hyperparameter values used in the molecular optimization experiments.

| Hyperparameter | Value |
|---|---|
| **ReLeaSE** | |
| Max. sequence length | 120 |
| Num. RNN layers | 2 |
| Num. MLP layers | 2 |
| Num. RNN and MLP hidden units | 128 |
| Batch size | 128 |
| Learning rate | 0.005 |
| Activation | ReLU |
| **REINVENT** | |
| Max. sequence length | 100 |
| Batch size | 128 |
| $\sigma$ [281] | 10 |
| Learning rate | 0.0005 |
| Num. GRU cells | 3 |
| Num. hidden units | 512 |
| **CDDD + PPO** | |
| PPO Num. steps | 1,024 |
| PPO Num. minibatches | 4 |
| PPO Num. opt. epochs | 4 |
| PPO clip factor | 0.2 |
| MLP layers | 3 |
| MLP hidden units | 512 |
| Activation | ReLU |
| **GraphINVENT** | |
| Max. molecular size | 100 |
| Batch size | 64 |
| Epochs | 300 |
| Block size | 10k |
| $\sigma$ [265] | 20 |
| $\alpha$ [265] | 0.5 |
| Learning rate | 0.0001 |

# Appendix B

# Additional examples of RL algorithms represented as graphs

This section presents additional examples of representing Reinforcement Learning algorithms and loss functions as computational graphs. In all cases, the notation used follows the style introduced by Co-Reyes et al. [67].

## B.1 Vanilla Policy Gradient

Figure B-1 presents the graph representation of both the policy gradient and the value function gradient for a Vanilla Policy Gradient algorithm.

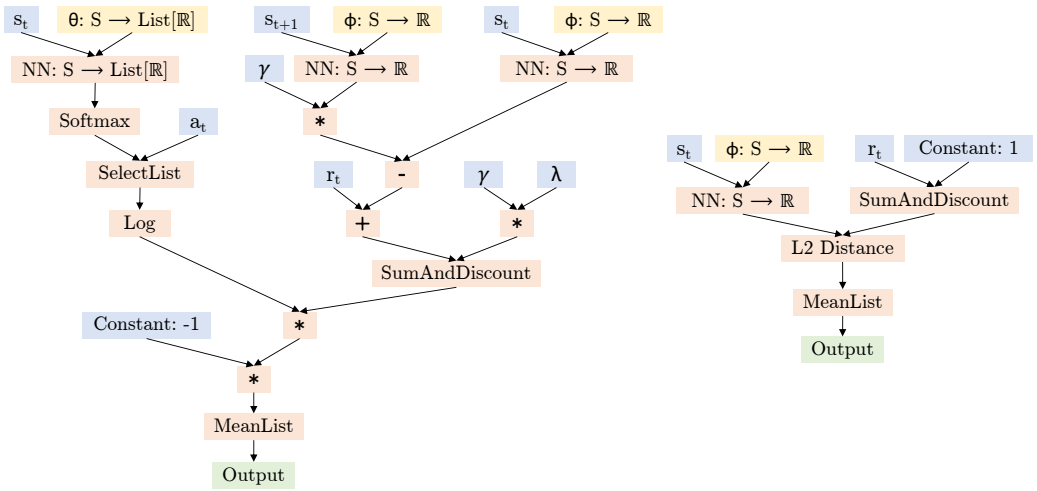



**Figure B-1:** Policy gradient (left) and value function gradient (right) in a Vanilla Policy Gradient algorithm represented as graphs.

## B.2 Deep Deterministic Policy Gradient (DDPG)

Figure B-2 shows the graph representation of both the Q-function gradient and the policy gradient for the Deep Deterministic Policy Gradient algorithm [233].

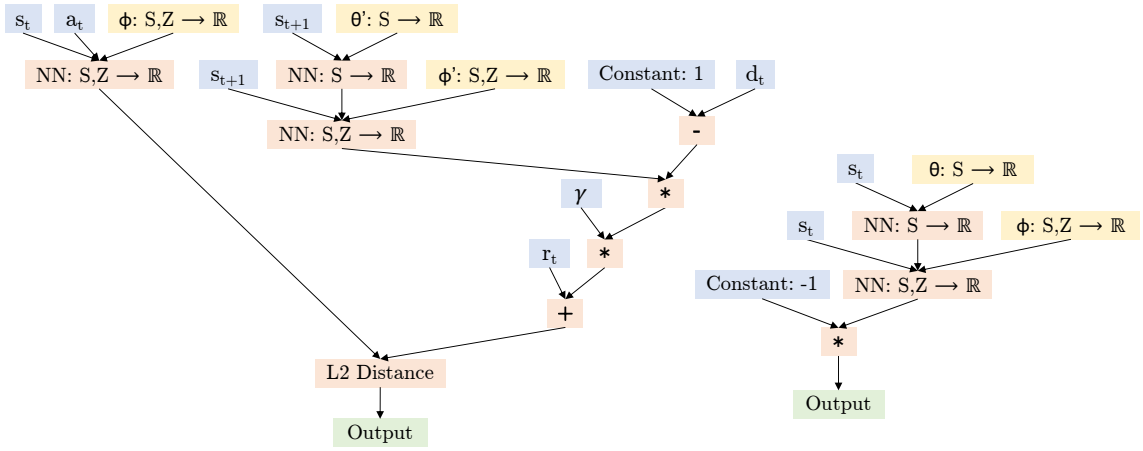



**Figure B-2:** Q-function gradient (left) and policy gradient (right) in the DDPG algorithm represented as graphs.

## B.3 Proximal Policy Optimization (PPO)

Figure B-3 presents the graph representation of the Proximal Policy Optimization loss function for the policy as presented in [328]. The value function gradient function is the same as the one introduced in the Vanilla Policy Gradient algorithm case.

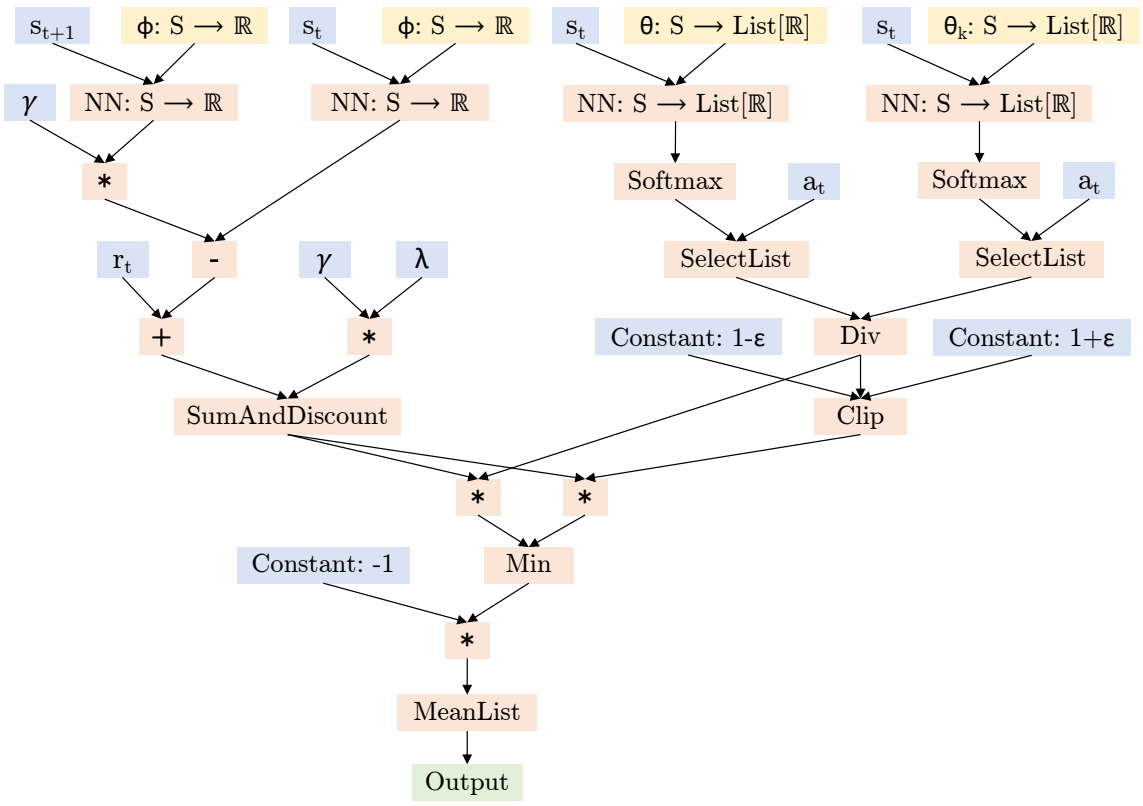



**Figure B-3:** Policy gradient of the PPO algorithm represented as a graph.

# Appendix C

# Additional evolution results

This section presents additional results of the evolution experiments carried out in Chapter 6. First, it includes PPO [328] in the performance and generalizability comparison for the case of RWRL Cartpole. Then, it presents the loss functions for the best performers and generalizers for each of the environments considered. Lastly, it shows the evolution of the best-in-population performance and generalizability.

## C.1 PPO

Figure C-1 extends Figure 6-3 presented in the evolution results for RWRL Cartpole. In addition, to the performance of the evolved algorithms and SAC, it also shows the performance of PPO, which has been hyperparameter-tuned for both performance and generalizability. In any case it is able to adequately complete the task.

## C.2 Evolved algorithms

This section presents the evolved loss functions (policy loss and critic loss) for the rest of the enviornments considered in this dissertation: RWRL Walker, OpenAI Gym Pendulum, Brax Ant, and Brax Humanoid.

### C.2.1 Best performer and best generalizer for RWRL Walker and OpenAI Gym Pendulum

This part presents the loss equations for both the best performer and best generalizer when using RWRL Walker and OpenAI Gym Pendulum as training environments.

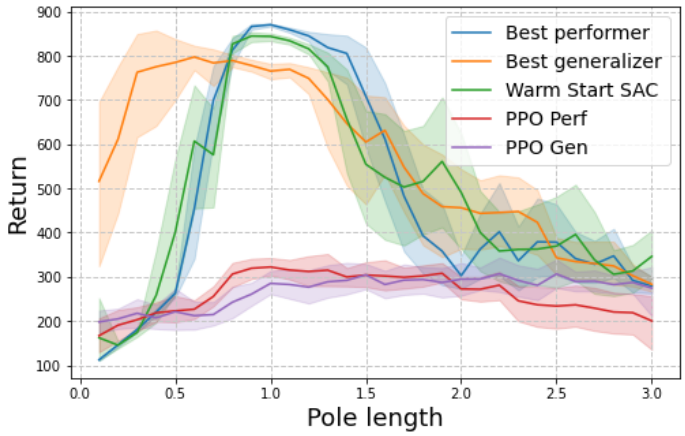



**Figure C-1:** Average and standard deviation across seeds of the meta-validation performance of the best performer, the best generalizer, the warm-start (SAC), and two hyperparameter-tuned versions of PPO (tuned for performance and generalizability, respectively) when training on a single configuration of RWRL Cartpole and evaluating on multiple unseen ones. The pole length changes across environment configurations and a length of 1.0 is used as training configuration.

First, the best performer for RWRL Walker:

$$L_{\pi}^{perf} = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[r_t + \gamma\left(\min_i Q_{targ_i}(s_{t+1},\tilde{a}_{t+1}) - \text{atan}(\gamma/Q(s_t,a_t))\right) - Q(s_t,\tilde{a}_t)\right] \tag{C.1}$$

$$L_{Q_i}^{perf} = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left(r_t + \gamma\left(\min_i Q_{targ_i}(s_{t+1},\tilde{a}_{t+1}) - \text{atan}(\gamma/Q_i(s_t,a_t))\right) - Q_i(s_t,a_t)\right)^2\right] \tag{C.2}$$

In all cases, $\tilde{a}_t \sim \pi(\cdot|s_t)$, $\tilde{a}_{t+1} \sim \pi(\cdot|s_{t+1})$, and $\mathcal{D}$ is a dataset of experience tuples from the replay buffer. Next, the best generalizer for RWRL Walker:

$$L_{\pi}^{gen} = \mathbb{E}_{(s_t,a_t,s_{t+1})\sim\mathcal{D}}\left[\frac{0.2\cdot\log\pi(\tilde{a}_{t+1}|s_{t+1})}{Q_i(s_{t+1},\tilde{a}_{t+1}) - 0.1\cdot\log\pi(\tilde{a}_{t+1}|s_{t+1})} - \min_i Q_i(s_t,\tilde{a}_{t+1})\right] \tag{C.3}$$

$$L_{Q_i}^{gen} = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left(r_t + \gamma\left(Q_i(s_{t+1},\tilde{a}_{t+1}) - 0.1\cdot\log\pi(\tilde{a}_{t+1}|s_{t+1})\right) - Q_i(s_t,a_t)\right)^2\right] \tag{C.4}$$

The best performer for Gym Pendulum is given by equations:

$$L_{\pi}^{perf} = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[2 \cdot \operatorname{atan}(\log \pi(\tilde{a}_t|s_t)) - \min_i Q_i(s_t, \tilde{a}_t)\right] \tag{C.5}$$

$$L_{Q_i}^{perf} = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left(r_t + \gamma\left(Q_{targ_i}(s_{t+1}, \tilde{a}_t) - \log \pi(\tilde{a}_t|s_t)\right) - Q_i(s_t, a_t)\right)^2\right] \tag{C.6}$$

Finally, the equations for the best generalizer when using Gym Pendulum are:

$$L_{\pi}^{gen} = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\log(\log \pi(\tilde{a}_t|s_t)) - \min_i Q_i(s_t, \tilde{a}_t)\right] \tag{C.7}$$

$$L_{Q_i}^{gen} = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left(r_t + \gamma\left(Q_{targ_i}(s_{t+1}, \tilde{a}_t) - \log(\log \pi(\tilde{a}_t|s_t))\right) - Q_i(s_t, a_t)\right)^2\right] \tag{C.8}$$

### C.2.2 Best performer for Brax Ant and Brax Humanoid

Lastly, the loss equations for the best performer algorithms evolved in Brax Ant and Brax Humanoid are presented; the analytical sections of this dissertation have focused on these two algorithms. The loss functions for the best performer for Brax Ant are:

$$L_{\pi}^{perf} = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\log \pi(\tilde{a}_{t+1}|s_{t+1}) - \min_i Q_i(s_t, a_t)\right] \tag{C.9}$$

$$L_{Q_i}^{perf} = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left|\left(r + \gamma\left(\min_i Q_{targ_i}(s_t, \tilde{a}_{t+1}) - \gamma\right) - Q_i(s_t, a_t)\right)^2 \cdot C_1\right|\right] \tag{C.10}$$

where

$$C_1 = r_t + \gamma \cdot \left(\min_i Q_i(s_{t+1}, \tilde{a}_{t+1}) - \gamma\right) \tag{C.11}$$

In all cases, $\tilde{a}_{t+1} \sim \pi(\cdot|s_{t+1})$ and $\mathcal{D}$ is a dataset of experience tuples from the replay buffer. Then, the equations for the best performer evolved in Brax Humanoid are:

$$L_{\pi}^{perf} = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\log \pi(\tilde{a}_{t+1}|s_{t+1}) - \min_i Q_i(s_t, a_t)\right] \tag{C.12}$$

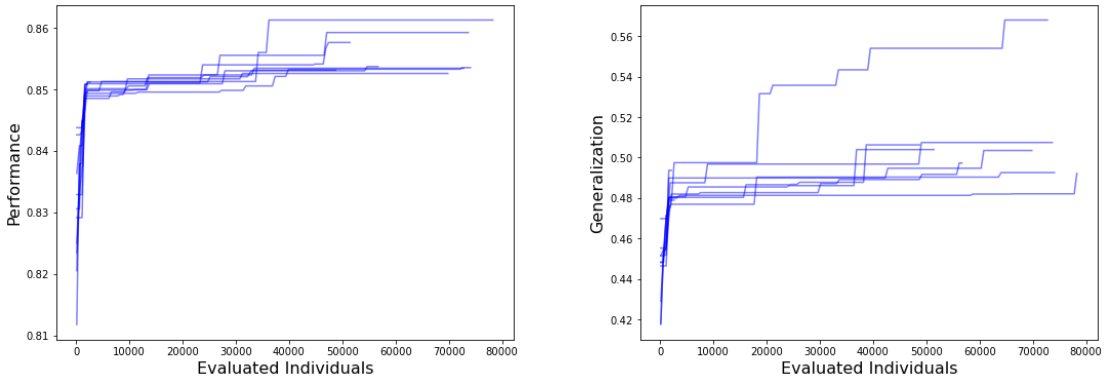
$$L_{Q_i}^{perf} = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left(C_2 - Q_i(s_t, a_t)\right)^2 \cdot C_2\right] \tag{C.13}$$

where

$$C_2 = r_t + \gamma\left(\min_i Q_{targ_i}(s_t, \tilde{a}_{t+1}) - \log \pi(\tilde{a}_{t+1}|s_{t+1})\right) \tag{C.14}$$

## C.3 Computational cost vs. performance

Figure C-2 shows the evolution of the best fitness in the population with respect to the total number of graphs evolved in the case of RWRL Cartpole; 10 independent runs of the same experiment are shown. It can be observed that the largest jumps in stability-adjusted performance and generalizability occur in the first 10% and 25% of individual evaluations, respectively. The stochastic nature of an individual experiment can lead to different outcomes with shorter or longer distances between fitness jumps. Aggregating populations —after meta-training— from 10 different experiments is a good way of countering such stochasticity.




**(a)** Stability-adjusted performance. **(b)** Stability-adjusted generalizability.

**Figure C-2:** Evolution curves of the best fitness in the population with respect to the total number of evaluated individuals. Figures show 10 identical runs of the evolution experiment using RWRL Cartpole.

# Appendix D

# Additional information use case satellite communications

Chapter 8 has introduced an Integer Linear Programming (ILP) formulation for the frequency assignment problem in satellite constellations. The variables and constraints have served to define a basis for the subsequent DRL system. Two elements remained to be discussed with respect to the ILP formulation: the objective function and the computational complexity. They are both addressed in this section.

## D.1 Objective function

The ideas presented in Chapter 8 include the essential constraints and decisions that contribute to the creation of *valid* frequency plans, which essentially refers to plans that conform to all existing constraints. But to make a selection between various valid frequency plans, an objective function is required. There may be situations where operators would favor plans that amplify bandwidth allocation or those that minimize frequency reuses. To integrate this adaptability into the ILP structure, the following objective function is proposed:

$$\max \sum_{i=1}^{N_B} (\beta_{1,i} b_i - |\beta_{2,i}| g_i - |\beta_{3,i}| f_i - |\beta_{4,i}| P_i(f_i, b_i)) \tag{D.1}$$

Here, $\beta_{1,i}$, $\beta_{2,i}$, $\beta_{3,i}$, and $\beta_{4,i}$ act as weighting parameters for beam $i$, with each $\beta_{k,i}$ being a real number. This function merges four distinct objectives:

1. $\beta_{1,i} b_i$ maximizes or minimizes (when $\beta_{1,i} < 0$) the bandwidth allocated to beam $i$. Having superior control over bandwidth utilization can lead to enhanced

power efficiency.

2. $-|\beta_{2,i}|g_i$ aims to minimize frequency reuses, true when $|\beta_{2,i}| > 0$. Setting $\beta_{2,i} = 0$ for all beams allows for a consistent use of frequency reuses and polarizations. Absolute terms for the coefficient $\beta_{2,i}$ are used because maximizing and minimizing $g_i$ have equivalent effects, as per the formulation introduced in Chapter 8.

3. $-|\beta_{3,i}|f_i$ aims to regulate the number of frequency slots used. If $-|\beta_{3,i}| < 0$, the lower parts of the spectrum are given precedence. If $\beta_{3,i} = 0$, the spectrum usage is even. This term is deducted given that lower frequencies may need less power and hence are occasionally preferred by operators.

4. When available, $-|\beta_{4,i}|P_i(f_i, b_i)$ serves as a substitute for the RF power consumption of beam $i$ for the slots $f_i, f_i + 1, ..., f_i + b_i - 1$. This operand signifies the preference to reduce RF power whenever feasible, when $\beta_{4,i} > 0$.

These weighting parameters serve to establish a priority order among these objectives. Though these parameters can be the same for all beams, operators might want to prioritize extra bandwidth or certain bands for particular beams.

## D.2 Computational complexity

Constructing a frequency plan is a problem acknowledged as NP-hard, a fact established through its association with the graph coloring problem, a subject of extensive research in graph theory [200]. In particular, under the assumption that each beam is assigned an equal bandwidth amount of 1 ($b_i = 1, \forall i$) and there exists solely a single frequency reuse ($N_{FR} = 1$), the problem's constraints turn into:

$$f_i \in \{1, ..., N_{BW}N_P\}, \qquad \forall\, i \in \{1, ..., N_B\} \tag{D.2}$$

$$f_i \neq f_j \;\forall\; \{i, j\} \in \mathcal{R}_A \cup \mathcal{R}_E \cup \mathcal{R}_G \tag{D.3}$$

This scenario translates into a solution space mirroring the graph coloring problem, a known NP-complete problem, as substantiated by preceding studies in the domain of spectrum management [49]. Navigating this solution space demands complexity of $\mathcal{O}((N_{BW}N_P)^{N_B})$. Reverting to the initial problem, with multiple bandwidths and frequency reuses, results in a solution space of order $\mathcal{O}((N_{BW}^2 N_P N_{FR})^{N_B})$.

To provide a frame of reference, a system comprising 5 potential bandwidths ($N_{BW}$), 2 polarizations ($N_P$), 2 frequency reuses ($N_{FR}$), and 1,000 beams ($N_B$) presents a solution space of magnitude $10^{2000}$. This implies that addressing the problem for a high quantity of beams becomes computationally unmanageable.

# Appendix E

# Additional experiments Traffic Signal Control

This section presents additional experiments for the Traffic Signal Control (TSC) use case introduced in Chapter 4. Specifically, the change in MI after removing features and the MI between pairs of features are discussed.

## E.1 The effect of removing features

The experiments in Section 4.6.2 have identified the phase, the total time waited in the intersection, and the number of cars waiting as the features that are statistically significant. This part examines what occurs in terms of learning and changes in mutual information (MI) 1) when those features are removed and 2) when those are the only features kept. Figure E-1 shows the change in MI across 100 episodes when masking the relevant features in the 7-intersection use case. The figure shows the MI remains constant at zero for all features, as the agent is not able to pick up any learning signal. In the baseline case, the agent identifies two important features that help getting information on the reward, then it progressively exploits those features until they are no longer informative —once the policy converges. Consequently, in practical applications, observing constant MI at zero for a sequence of episodes or epochs might indicate more features are needed, as the agent cannot learn from any. Note that when looking at a slice of the learning process, one could observe zero or close to zero MI for all features without indicating the agent struggles to learn a policy, this would be the observation in the last episodes in Figures 4-7 and 4-8.

Then, Figure E-2 shows the same procedure when masking all variables except

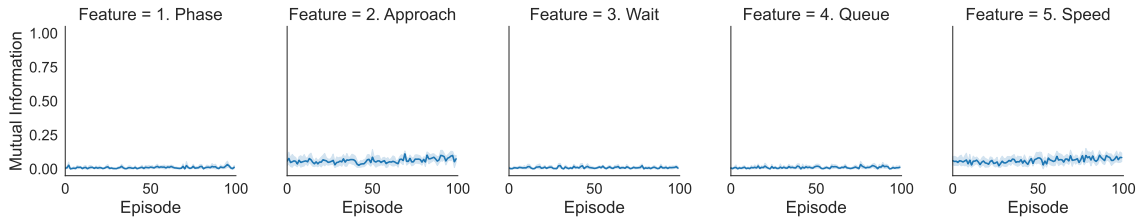



**Figure E-1:** Masking the phase, the total waiting time, and the number of vehicles waiting in the 7-intersection use case. Evolution of the Mutual Information between the reward and each of the five features considered in the RESCO benchmark for Traffic Signal Control during learning. The mean and 95% CI across 7 intersections is shown.

from the three significant ones. In this case, since these are the only features the agent needs to learn the policy, the MI curve is similar to the one observed in the baseline case. Observing negligible MI in useless features is something that agrees with other studies [247].

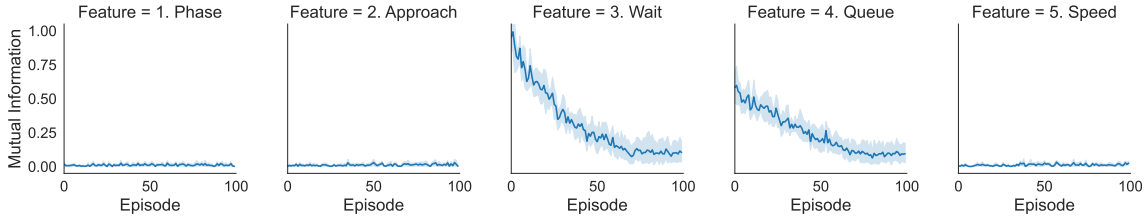



**Figure E-2:** Masking the number of vehicles approaching and the average speed in the 7-intersection use case. Evolution of the Mutual Information between the reward and each of the five features considered in the RESCO benchmark for Traffic Signal Control during learning. The mean and 95% CI across 7 intersections is shown.

## E.2 Comparing the mutual information between features

Finally, this section complements the results on MI between state features and rewards by examining the MI between pairs of state features as the agent learns the policy. This experiment shows that the MI between pairs of features also changes as the agent learns the policy. Figure E-3 shows the change in MI between pairs of features as the policy converges in the 7-intersection use case.

The experiments in Chapter 4 have demonstrated that the total waited time at the intersection (feature 3) and the number of cars in the queue (feature 4) are both

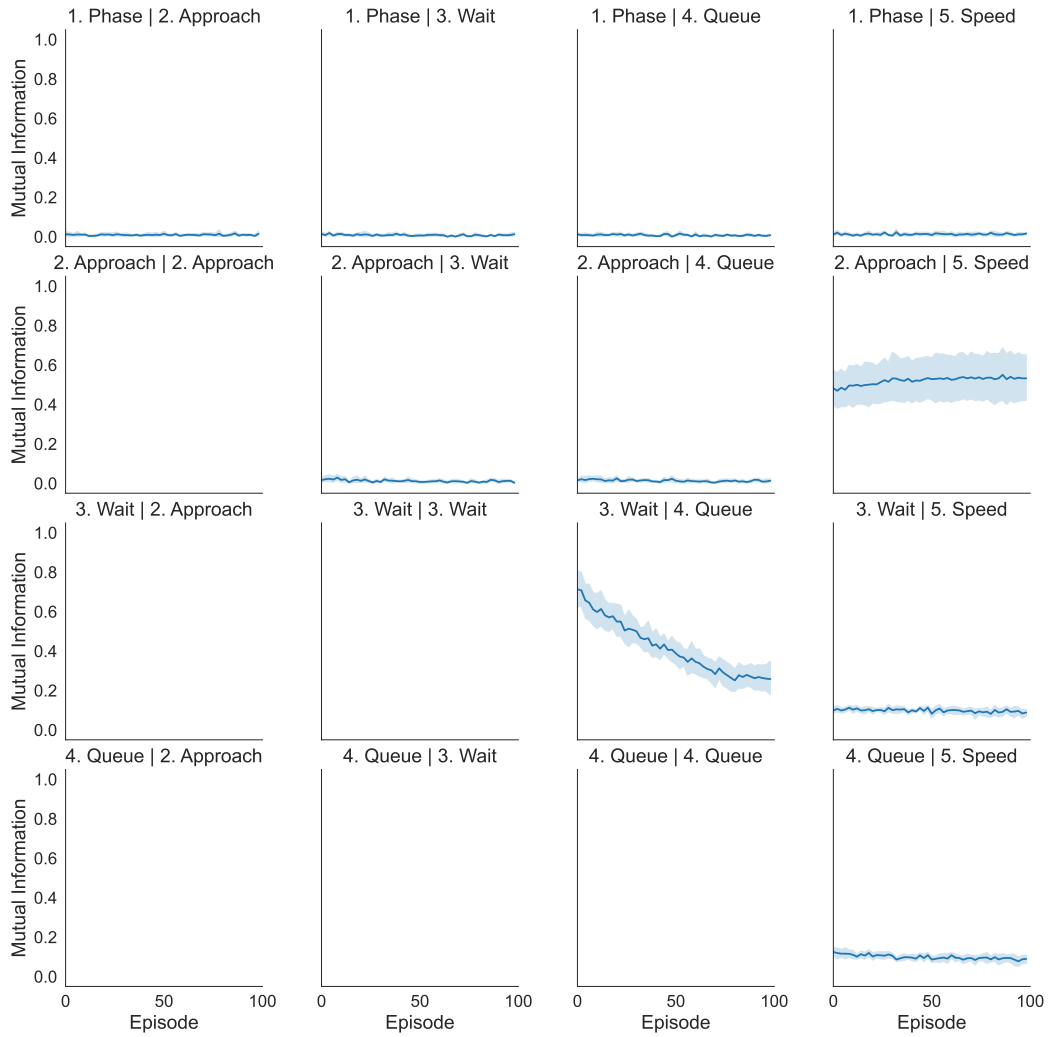



**Figure E-3:** Evolution in the pairwise Mutual Information between features during learning in the 7-intersection use case. Each row and column correspond to a different feature and only the upper side and the diagonal of the grid are relevant. The title of each cell indicates which two features are being considered. The mean and 95% CI across 7 intersections is shown.

relevant features and their MI with respect to the reward is high at the beginning of learning and low at the end. This is also observed when examining the MI between these two features, although the MI does not drop to zero when the policy converges, indicating there is some degree of redundancy in the information provided by both features at the moment of convergence. At the beginning of learning, both features have a high MI, indicating that only one of them might be enough. However, as the agent learns, both become relevant separately. This observation is interesting to better understand whether relevant features are relevant at all times during learning or if their relevance changes as the policy changes.

Then, the baseline experiments have also showed that the number of cars approaching (feature 2) and the average speed in the lane (feature 5) are not necessary features. These figures show that there is high redundancy given by moderately high MI between these two features during the whole learning process —it even slightly increases. While this indicates that at least one of them is not necessary, in the end both features are not statistically significant. Figure E-4 displays the same behaviors in the 21-intersection use case.

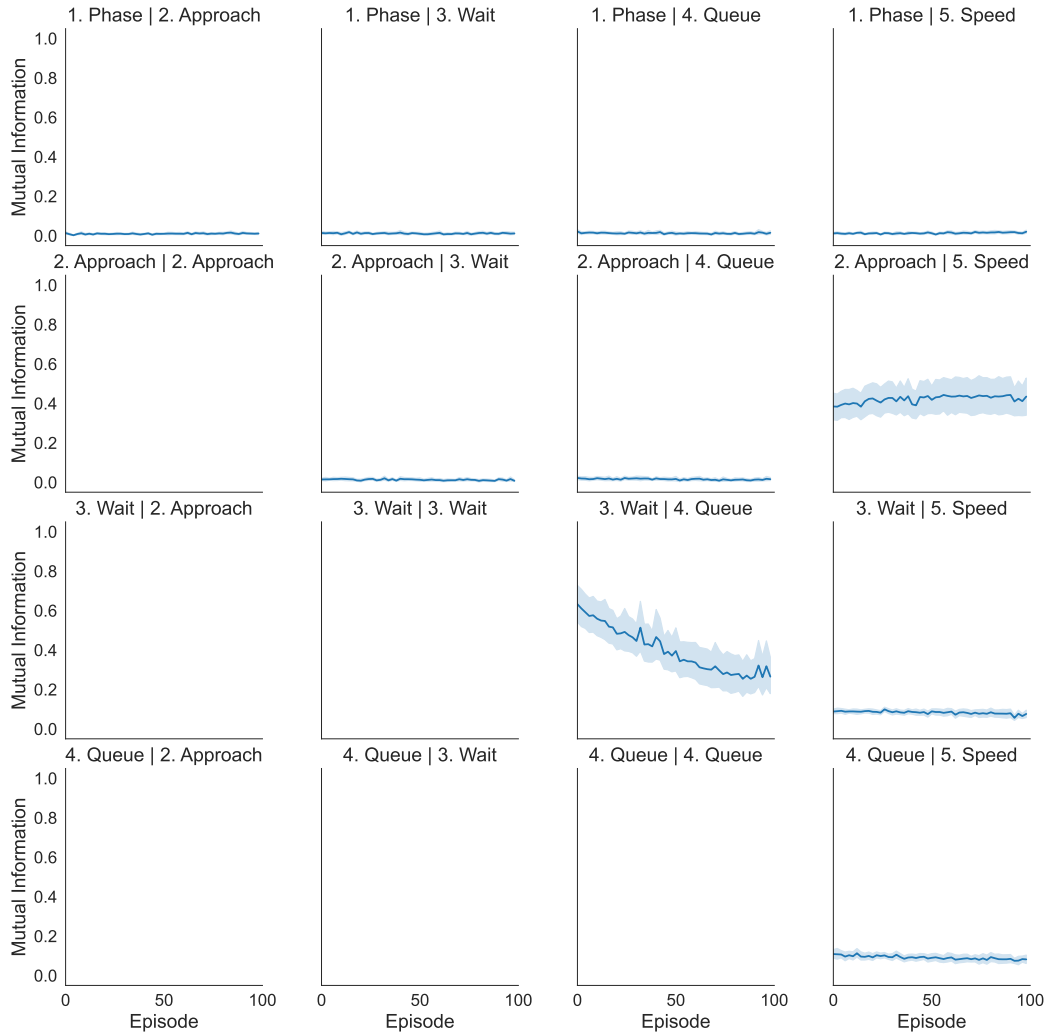



**Figure E-4:** Evolution in the pairwise Mutual Information between features during learning in the 21-intersection use case. Each row and column correspond to a different feature and only the upper side and the diagonal of the grid are relevant. The title of each cell indicates which two features are being considered. The mean and 95% CI across 21 intersections is shown.

# Bibliography


[1] Nadine Abbas, Youssef Nasser, and Karim El Ahmad. Recent advances on artificial intelligence and learning techniques in cognitive radio networks. *Eurasip Journal on Wireless Communications and Networking*, 2015(1), 2015.

[2] Abbas Abdolmaleki, Sandy Huang, Leonard Hasenclever, Michael Neunert, Francis Song, Martina Zambelli, Murilo Martins, Nicolas Heess, Raia Hadsell, and Martin Riedmiller. A distributional view on multi-objective policy optimization. In *International Conference on Machine Learning*, pages 11–22. PMLR, 2020.

[3] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a Posteriori Policy Optimisation. jun 2018.

[4] Tedros Salih Abdu, Steven Kisseleff, Eva Lagunas, and Symeon Chatzinotas. Flexible resource optimization for geo multibeam satellite communication system. *IEEE Transactions on Wireless Communications*, 20(12):7888–7902, 2021.

[5] Tedros Salih Abdu, Steven Kisseleff, Eva Lagunas, Symeon Chatzinotas, and Björn Ottersten. Demand and interference aware adaptive resource management for high throughput geo satellite systems. *IEEE Open Journal of the Communications Society*, 3:759–775, 2022.

[6] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[7] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31. PMLR, 2017.

[8] Igor Adamski, Robert Adamski, Tomasz Grel, Adam Jędrych, Kamil Kaczmarek, and Henryk Michalewski. Distributed Deep Reinforcement Learning: Learn How to Play Atari Games in 21 minutes. pages 370–388. 2018.

[9] M. Mehdi Afsar, Trafford Crump, and Behrouz Far. Reinforcement learning based recommender systems: A survey. jan 2021.

[10] Reza Refaei Afshar, Yingqian Zhang, Murat Firat, and Uzay Kaymak. A state aggregation approach for solving knapsack problem with deep reinforcement learning. In *Asian Conference on Machine Learning*, pages 81–96. PMLR, 2020.

[11] Rishabh Agarwal, Marlos C Machado, Pablo Samuel Castro, and Marc G Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. *arXiv preprint arXiv:2101.05265*, 2021.

[12] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[13] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

[14] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[15] Ferran Alet, Martin F Schneider, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Meta-learning curiosity algorithms. In *International Conference on Learning Representations*, 2019.

[16] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

[17] Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*, pages 13566–13577, 2019.

[18] Mohammad Aslani, Mohammad Saadi Mesgari, Stefan Seipel, and Marco Wiering. Developing adaptive traffic signal control by actor–critic and direct exploration methods. In *Proceedings of the Institution of Civil Engineers-Transport*, volume 172, pages 289–298. Thomas Telford Ltd, 2019.

[19] Sara Romeo Atance, Juan Viguera Diez, Ola Engkvist, Simon Olsson, and Rocío Mercado. De novo drug design using reinforcement learning with graph-based deep generative models. *Journal of Chemical Information and Modeling*, 62(20):4863–4872, 2022.

[20] James Ault and Guni Sharon. Reinforcement learning benchmarks for traffic signal control. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[21] Ahmad Taher Azar, Anis Koubaa, Nada Ali Mohamed, Habiba A Ibrahim, Zahra Fathy Ibrahim, Muhammad Kazim, Adel Ammar, Bilel Benjdira, Alaa M Khamis, Ibrahim A Hameed, et al. Drone deep reinforcement learning: A review. *Electronics*, 10(9):999, 2021.

[22] Cédric Balty, Jean-Didier Gayrard, and Patrick Agnieray. Communication satellites to enter a new age of flexibility. *Acta Astronautica*, 65(1-2):75–81, jul 2009.

[23] Yaoyao Bao, Yuanming Zhu, and Feng Qian. A deep reinforcement learning approach to improve the learning performance in process control. *Industrial & Engineering Chemistry Research*, 60(15):5504–5515, 2021.

[24] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed Distributional Deterministic Policy Gradients. apr 2018.

[25] Andrew G Barto and Richard S Sutton. Goal seeking components for adaptive intelligence: An initial assessment. Technical report, MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER AND INFORMATION SCIENCE, 1981.

[26] Andrew G Barto and Richard S Sutton. Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element. *Behavioural Brain Research*, 4(3):221–235, 1982.

[27] Andrew G Barto, Richard S Sutton, and Peter S Brouwer. Associative search network: A reinforcement learning associative memory. *Biological cybernetics*, 40:201–211, 1981.

[28] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. DeepMind Lab. dec 2016.

[29] Sarah Bechtle, Artem Molchanov, Yevgen Chebotar, Edward Grefenstette, Ludovic Righetti, Gaurav Sukhatme, and Franziska Meier. Meta Learning via Learned Loss. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168. IEEE, jan 2021.

[30] 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, jun 2013.

[31] Marc G. Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C. Machado, Subhodeep Moitra, Sameera S. Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, dec 2020.

[32] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[33] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[34] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[35] Felix Berkenkamp, Matteo Turchetta, Angela P Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. *arXiv preprint arXiv:1705.08551*, 2017.

[36] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific, 2012.

[37] Dimitri P Bertsekas, David A Castanon, et al. Adaptive aggregation methods for infinite horizon dynamic programming. 1988.

[38] Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[39] A. Birhane, A. Kasirzadeh, D. Leslie, and et al. Science in the age of large language models. *Nat Rev Phys*, 5:277–280, 2023.

[40] Steven Bohez, Abbas Abdolmaleki, Michael Neunert, Jonas Buchli, Nicolas Heess, and Raia Hadsell. Value constrained model-free continuous control. feb 2019.

[41] Matthew Michael Botvinick. Hierarchical reinforcement learning and decision making. *Current opinion in neurobiology*, 22(6):956–962, 2012.

[42] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[43] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[44] Johannes Broichhagen, James Allen Frank, and Dirk Trauner. A roadmap to success in photopharmacology. *Accounts of chemical research*, 48(7):1947–1960, 2015.

[45] Nathan Brown, Marco Fiscato, Marwin H.S. Segler, and Alain C. Vaucher. GuacaMol: Benchmarking Models for de Novo Molecular Design. *Journal of Chemical Information and Modeling*, 59(3):1096–1108, mar 2019.

[46] Noam Brown and Tuomas Sandholm. Solving imperfect-information games via discounted regret minimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1829–1836, 2019.

[47] Arthur E Bryson. Optimal control-1950 to 1985. *IEEE Control Systems Magazine*, 16(3):26–33, 1996.

[48] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pages 8224–8234, 2018.

[49] Jean-Thomas Camino, Stephane Mourgues, Christian Artigues, and Laurent Houssin. A greedy approach combined with graph coloring for non-uniform beam layouts under antenna constraints in multibeam satellite systems. In *2014 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, pages 374–381. IEEE, sep 2014.

[50] Anthony R Cassandra, Leslie Pack Kaelbling, and Michael L Littman. Acting optimally in partially observable stochastic domains. In *Aaai*, volume 94, pages 1023–1028, 1994.

[51] Adrià Cereto-Massagué, María José Ojeda, Cristina Valls, Miquel Mulero, Santiago Garcia-Vallvé, and Gerard Pujadas. Molecular fingerprint similarity search in virtual screening. *Methods*, 71:58–63, 2015.

[52] Johan Samir Obando Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *International Conference on Machine Learning*, pages 1373–1383. PMLR, 2021.

[53] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient Lifelong Learning with A-GEM. dec 2018.

[54] Baiming Chen, Mengdi Xu, Liang Li, and Ding Zhao. Delay-Aware Model-Based Reinforcement Learning for Continuous Control. may 2020.

[55] Irene Y Chen, Emma Pierson, Sherri Rose, Shalmali Joshi, Kadija Ferryman, and Marzyeh Ghassemi. Ethical machine learning in healthcare. *Annual review of biomedical data science*, 4:123–144, 2021.

[56] Jerry Zikun Chen. Reinforcement learning generalization with surprise minimization. *arXiv preprint arXiv:2004.12399*, 2020.

[57] Peng Chen, Zemao Zhu, and Guangquan Lu. An adaptive control method for arterial signal coordination based on deep reinforcement learning. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3553–3558. IEEE, 2019.

[58] Sikai Chen, Jiqian Dong, Paul Ha, Yujie Li, and Samuel Labi. Graph neural network and reinforcement learning for multi-agent cooperative control of

connected autonomous vehicles. *Computer-Aided Civil and Infrastructure Engineering*, 36(7):838–857, 2021.

[59] Xi Chen, Ali Ghadirzadeh, Marten Bjorkman, and Patric Jensfelt. Meta-Learning for Multi-objective Reinforcement Learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 977–983. IEEE, nov 2019.

[60] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning Navigation Behaviors End-to-End With AutoRL. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, apr 2019.

[61] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, April 2019.

[62] J P Choi and V W S Chan. Optimum power and beam allocation based on traffic demands and channel conditions over satellite downlinks. *IEEE Transactions on Wireless Communications*, 4(6):2983–2993, nov 2005.

[63] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A Lyapunov-based Approach to Safe Reinforcement Learning. may 2018.

[64] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[65] Tianshu Chu, Jie Wang, Lara Codecà, and Zhaojian Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):1086–1095, 2019.

[66] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

[67] John D Co-Reyes, Yingjie Miao, Daiyi Peng, Esteban Real, Sergey Levine, Quoc V Le, Honglak Lee, and Aleksandra Faust. Evolving reinforcement learning algorithms. In *International Conference on Learning Representations*, 2021.

[68] John D. Co-Reyes, Yingjie Miao, Daiyi Peng, Esteban Real, Sergey Levine, Quoc V. Le, Honglak Lee, and Aleksandra Faust. Evolving Reinforcement Learning Algorithms. jan 2021.

[69] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[70] Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *International Conference on Machine Learning*, pages 2020–2027. PMLR, 2021.

[71] Giuseppe Cocco, Tomaso De Cola, Martina Angelone, Zoltan Katona, and Stefan Erl. Radio Resource Management Optimization of Flexible Satellite Payloads for DVB-S2 Systems. *IEEE Transactions on Broadcasting*, 64(2):266–280, 2018.

[72] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. How many random seeds? statistical power analysis in deep reinforcement learning experiments. *arXiv preprint arXiv:1806.08295*, 2018.

[73] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms. *arXiv preprint arXiv:1904.06979*, 2019.

[74] Martin Coleman. Is AI key to the survival of satcoms?, 2019.

[75] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[76] Edward Crawley, Bruce Cameron, and Daniel Selva. *System architecture: Strategy and product development for complex systems*. Prentice Hall Press, 2015.

[77] Mary Cummings. Informing autonomous system design through the lens of skill-, rule-, and knowledge-based behaviors. *Journal of Cognitive Engineering and Decision Making*, 12(1):58–61, 2018.

[78] Mary L Cummings and Songpo Li. Subjectivity in the creation of machine learning models. *ACM Journal of Data and Information Quality*, 13(2):1–19, 2021.

[79] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pages 1096–1105. PMLR, 2018.

[80] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

[81] Suresh Dara, Swetha Dhamercherla, Surender Singh Jadav, CH Madhu Babu, and Mohamed Jawed Ahsan. Machine learning in drug discovery: a review. *Artificial Intelligence Review*, 55(3):1947–1999, 2022.

[82] Olivier L De Weck. *Technology Roadmapping and Development: A Quantitative Approach to the Management of Technology*. Springer Nature, 2022.

[83] Tom de Wiele, David Warde-Farley, Andriy Mnih, and Volodymyr Mnih. Q-Learning in enormous action spaces via amortized approximate maximization. *arXiv preprint arXiv:2001.08116*, 2020.

[84] Christian Schroeder de Witt, Bei Peng, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, and Shimon Whiteson. Deep Multi-Agent Reinforcement Learning for Decentralized Continuous Cooperative Control. mar 2020.

[85] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[86] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, feb 2022.

[87] Deloitte. Business impacts of machine learning. Technical report, 2017.

[88] Boyu Deng, Chunxiao Jiang, Haipeng Yao, Song Guo, and Shanghong Zhao. The Next Generation Heterogeneous Satellite Communication Networks: Integration of Resource Management and Deep Reinforcement Learning. *IEEE Wireless Communications*, 27(2):105–111, apr 2020.

[89] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre M. Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[90] Esther Derman, Gal Dalal, and Shie Mannor. Acting in Delayed Environments with Non-Stationary Markov Policies. jan 2021.

[91] Esther Derman, Daniel Mankowitz, Timothy Mann, and Shie Mannor. A bayesian approach to robust reinforcement learning. In *Uncertainty in Artificial Intelligence*, pages 648–658. PMLR, 2020.

[92] Esther Derman, Daniel J. Mankowitz, Timothy A. Mann, and Shie Mannor. Soft-Robust Actor-Critic Policy-Gradient. mar 2018.

[93] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. OpenAI Baselines. https://github.com/openai/baselines, 2017.

[94] Matthias Dorfer, Anton R Fuxjäger, Kristian Kozak, Patrick M Blies, and Marcel Wasserer. Power grid congestion management via topology optimization with alphazero. *arXiv preprint arXiv:2211.05612*, 2022.

[95] Dominique Douguet, Etienne Thoreau, and Gérard Grassy. A genetic algorithm for the automated generation of small organic molecules: drug design using an evolutionary algorithm. *Journal of computer-aided molecular design*, 14:449–466, 2000.

[96] Iddo Drori, Anant Kharkar, William R. Sickinger, Brandon Kates, Qiang Ma, Suwen Ge, Eden Dolev, Brenda Dietrich, David P. Williamson, and Madeleine Udell. Learning to Solve Combinatorial Optimization Problems on Real-World Graphs in Linear Time. jun 2020.

[97] Yuanqi Du, Tianfan Fu, Jimeng Sun, and Shengchao Liu. Molgensurvey: A systematic survey in machine learning models for molecule design. *arXiv preprint arXiv:2203.14500*, 2022.

[98] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017.

[99] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep Reinforcement Learning in Large Discrete Action Spaces. dec 2015.

[100] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, pages 1–50, 2021.

[101] Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, apr 2021.

[102] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of Real-World Reinforcement Learning. *arXiv preprint arXiv:1904.12901*, 2019.

[103] Tyna Eloundou, Sam Manning, Pamela Mishkin, and Daniel Rock. Gpts are gpts: An early look at the labor market impact potential of large language models. *arXiv preprint arXiv:2303.10130*, 2023.

[104] Daniel C. Elton, Zois Boukouvalas, Mark D. Fuge, and Peter W. Chung. Deep learning for molecular design—a review of the state of the art. *Molecular Systems Design & Engineering*, 4(4):828–849, 2019.

[105] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International conference on learning representations*, 2019.

[106] Myungeun Eom and Byung-In Kim. The traffic signal control problem for intersections: a review. *European transport research review*, 12(1):1–20, 2020.

[107] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. feb 2018.

[108] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature Medicine*, 25(1):24–29, jan 2019.

[109] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is All You Need: Learning Skills without a Reward Function. feb 2018.

[110] Aleksandra Faust, Anthony Francis, and Dar Mehta. Evolving rewards to automate reinforcement learning. In *6th ICML Workshop on Automated Machine Learning*, 2019.

[111] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

[112] Fabio Ferreira, Thomas Nierhoff, and Frank Hutter. Learning synthetic environments for reinforcement learning with evolution strategies. *arXiv preprint arXiv:2101.09721*, 2021.

[113] Paulo Victor R. Ferreira, Randy Paffenroth, Alexander M. Wyglinski, Timothy M. Hackett, Sven G. Bilen, Richard C. Reinhart, and Dale J. Mortensen. Reinforcement Learning for Satellite Communications: From LEO to Deep Space Operations. *IEEE Communications Magazine*, 57(5):70–75, may 2019.

[114] Paulo Victor Rodrigues Ferreira, Randy Paffenroth, Alexander M. Wyglinski, Timothy M. Hackett, Sven G. Bilen, Richard C. Reinhart, and Dale J. Mortensen. Multiobjective Reinforcement Learning for Cognitive Satellite Communications Using Deep Neural Network Ensembles. *IEEE Journal on Selected Areas in Communications*, 36(5):1030–1041, 2018.

[115] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. mar 2017.

[116] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[117] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A Connection between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models. nov 2016.

[118] Chelsea Finn, Tianhe Yu, Justin Fu, Pieter Abbeel, and Sergey Levine. Generalizing Skills with Semi-Supervised Reinforcement Learning. dec 2016.

[119] Daniel Flam-Shepherd, Alexander Zhigalin, and Alán Aspuru-Guzik. Scalable fragment-based 3d molecular design with reinforcement learning. *arXiv preprint arXiv:2202.00658*, 2022.

[120] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1515–1528. PMLR, 10–15 Jul 2018.

[121] Luciano Floridi and Massimo Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694, 2020.

[122] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[123] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip H. S. Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning. feb 2017.

[124] G Fontanesi, F Ortíz, E Lagunas, V Monzon Baeza, MÁ Vázquez, JA Vásquez-Peralvo, M Minardi, HN Vu, PJ Honnaiah, C Lacoste, et al. Artificial intelligence for satellite communication and non-terrestrial networks: A survey. *arXiv preprint arXiv:2304.13008*, 2023.

[125] Roy Fox, Ari Pakman, and Naftali Tishby. Taming the noise in reinforcement learning via soft updates. *arXiv preprint arXiv:1512.08562*, 2015.

[126] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021.

[127] Justin Fu, Katie Luo, and Sergey Levine. Learning Robust Rewards with Adversarial Inverse Reinforcement Learning. oct 2017.

[128] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[129] Adam Gaier and David Ha. Weight agnostic neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[130] Robert Galvin. Science roadmaps. *Science*, 280(5365):803–804, 1998.

[131] Wenhao Gao, Tianfan Fu, Jimeng Sun, and Connor Coley. Sample efficiency matters: a benchmark for practical molecular optimization. *Advances in Neural Information Processing Systems*, 35:21342–21357, 2022.

[132] Juan Jose Garau-Luis, Edward Crawley, and Bruce Cameron. Applicability and Challenges of Deep Reinforcement Learning for Satellite Frequency Plan Design. In *2021 IEEE Aerospace Conference*, pages 1–11. IEEE, mar 2021.

[133] Juan Jose Garau Luis, Markus Guerster, Inigo del Portillo, Edward Crawley, and Bruce Cameron. Deep Reinforcement Learning Architecture for Continuous Power Allocation in High Throughput Satellites. *arXiv preprint arXiv:1906.00571*, jun 2019.

[134] Juan Jose Garau-Luis, Yingjie Miao, John D Co-Reyes, Aaron Parisi, Jie Tan, Esteban Real, and Aleksandra Faust. Evolving pareto-optimal actor-critic algorithms for generalizability and stability. *arXiv preprint arXiv:2204.04292*, 2022.

[135] Juan Jose Garau Luis, Nils Pachler, Markus Guerster, Inigo del Portillo, Edward Crawley, and Bruce Cameron. Artificial Intelligence Algorithms for Power Allocation in High Throughput Satellites: A Comparison. In *2020 IEEE Aerospace Conference*, pages 1–15. IEEE, mar 2020.

[136] Javier Garcia and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[137] Paul Garnier, Jonathan Viquerat, Jean Rabault, Aurélien Larcher, Alexander Kuhnle, and Elie Hachem. A review on deep reinforcement learning for fluid mechanics. *Computers & Fluids*, 225:104973, 2021.

[138] Paul Garnier, Jonathan Viquerat, Jean Rabault, Aurélien Larcher, Alexander Kuhnle, and Elie Hachem. A review on deep reinforcement learning for fluid mechanics. *Computers & Fluids*, 225:104973, 2021.

[139] Jason Gauci, Edoardo Conti, Yitao Liang, Kittipat Virochsiri, Yuchen He, Zachary Kaden, Vivek Narayanan, Xiaohui Ye, Zhengxing Chen, and Scott

Fujimoto. Horizon: Facebook's Open Source Applied Reinforcement Learning Platform, nov 2018.

[140] Hongwei Ge, Yumei Song, Chunguo Wu, Jiankang Ren, and Guozhen Tan. Cooperative deep q-learning with q-value transfer for multi-intersection signal control. *IEEE Access*, 7:40797–40809, 2019.

[141] Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-Conquer Reinforcement Learning. nov 2017.

[142] Xiaotian Gong, Lijuan Sun, Jian Zhou, Juan Wang, and Fu Xiao. Adaptive Routing Strategy Based on Improved Q-learning for Satellite Internet of Things. pages 161–172. 2021.

[143] Yaobang Gong, Mohamed Abdel-Aty, Qing Cai, and Md Sharikur Rahman. Decentralized network level adaptive signal control by multi-agent deep reinforcement learning. *Transportation Research Interdisciplinary Perspectives*, 1:100020, 2019.

[144] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[145] Sai Krishna Gottipati, Boris Sattarov, Sufeng Niu, Yashaswi Pathak, Haoran Wei, Shengchao Liu, Shengchao Liu, Simon Blackburn, Karam Thomas, Connor Coley, Jian Tang, Sarath Chandar, and Yoshua Bengio. Learning to Navigate The Synthetically Accessible Chemical Space Using Reinforcement Learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3668–3679. PMLR, 2020.

[146] Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models, 2018.

[147] Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gomez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, Gabriel Dulac-Arnold, Jerry Li, Mohammad Norouzi, Matt Hoffman, Ofir Nachum, George Tucker, Nicolas Heess, and Nando de Freitas. RL Unplugged: A Suite of Benchmarks for Offline Reinforcement Learning. jun 2020.

[148] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017.

[149] Izzeddin Gur, Natasha Jaques, Yingjie Miao, Jongwook Choi, Manoj Tiwari, Honglak Lee, and Aleksandra Faust. Environment generation for zero-shot compositional reinforcement learning. In Marc'Aurelio Ranzato, Alina Beygelzimer,

Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 4157–4169, 2021.

[150] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[151] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to Walk via Deep Reinforcement Learning. dec 2018.

[152] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[153] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications. dec 2018.

[154] Hirotaka Hachiya and Masashi Sugiyama. Feature selection for reinforcement learning: Evaluating implicit state-reward dependency via conditional mutual information. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part I 21*, pages 474–489. Springer Berlin Heidelberg, 2010.

[155] Patrick Halina, Mehdi Ben Ayed, Peng Zhong, and Curren Pangler. RL Bakery. https://github.com/zynga/rl-bakery, 2021.

[156] Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A. Efros, Lerrel Pinto, and Xiaolong Wang. Self-Supervised Policy Adaptation during Deployment. jul 2020.

[157] Botao Hao, Yaqi Duan, Tor Lattimore, Csaba Szepesvári, and Mengdi Wang. Sparse feature selection makes batch reinforcement learning more sample efficient. In *International Conference on Machine Learning*, pages 4063–4073. PMLR, 2021.

[158] Markus Hartenfeller, Ewgenij Proschak, Andreas Schüller, and Gisbert Schneider. Concept of combinatorial de novo design of drug-like molecules by particle swarm optimization. *Chemical biology & drug design*, 72(1):16–26, 2008.

[159] Matthew Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. jul 2015.

[160] Ammar Haydari and Yasin Yilmaz. Deep Reinforcement Learning for Intelligent Transportation Systems: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–22, 2020.

[161] Tairan He, Yuge Zhang, Kan Ren, Minghuan Liu, Che Wang, Weinan Zhang, Yuqing Yang, and Dongsheng Li. Reinforcement learning with automated auxiliary loss search. *arXiv preprint arXiv:2210.06041*, 2022.

[162] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018.

[163] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[164] Heng Wang, Aijun Liu, Xiaofei Pan, and Luliang Jia. Optimal bandwidth allocation for multi-spot-beam satellite communication systems. In *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, pages 2794–2798. IEEE, dec 2013.

[165] Lars Hertel, Pierre Baldi, and Daniel L Gillen. Quantity vs. quality: On hyperparameter optimization for deep reinforcement learning. *arXiv preprint arXiv:2007.14604*, 2020.

[166] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[167] Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. On Inductive Biases in Deep Reinforcement Learning. jul 2019.

[168] Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. On inductive biases in deep reinforcement learning. *arXiv preprint arXiv:1907.02908*, 2019.

[169] Carolina Higuera, Fernando Lozano, Edgar Camilo Camacho, and Carlos Hernando Higuera. Multiagent reinforcement learning applied to traffic light signal control. In *International conference on practical applications of agents and multi-agent systems*, pages 115–126. Springer, 2019.

[170] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 1997.

[171] Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

[172] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed Prioritized Experience Replay. mar 2018.

[173] Thanapapas Horsuwan and Chaodit Aswakul. Reinforcement learning agent under partial observability for traffic light control in presence of gridlocks. In *SUMO*, pages 29–47, 2019.

[174] Julien Horwood and Emmanuel Noutahi. Molecular design in synthetically accessible chemical space via deep reinforcement learning. *ACS omega*, 5(51):32984–32994, 2020.

[175] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-Learning in Neural Networks: A Survey. apr 2020.

[176] Rein Houthooft, Yuhua Chen, Phillip Isola, Bradly Stadie, Filip Wolski, OpenAI Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. In *Advances in Neural Information Processing Systems*, pages 5400–5409, 2018.

[177] Ronald A Howard. Dynamic programming and markov processes. 1960.

[178] Xin Hu, Xianglai Liao, Zhijun Liu, Shuaijun Liu, Xin Ding, Mohamed Helaoui, Weidong Wang, and Fadhel M. Ghannouchi. Multi-agent deep reinforcement learning-based flexible satellite payload for mobile terminals. *IEEE Transactions on Vehicular Technology*, 69(9):9849–9865, 2020.

[179] Xin Hu, Shuaijun Liu, Rong Chen, Weidong Wang, and Chunting Wang. A Deep Reinforcement Learning-Based Framework for Dynamic Resource Allocation in Multibeam Satellite Systems. *IEEE Communications Letters*, 22(8):1612–1615, 2018.

[180] Xin Hu, Yuchen Zhang, Xianglai Liao, Zhijun Liu, Weidong Wang, and Fadhel M. Ghannouchi. Dynamic Beam Hopping Method Based on Multi-Objective Deep Reinforcement Learning for Next Generation Satellite Broadband Systems. *IEEE Transactions on Broadcasting*, 66(3):630–646, sep 2020.

[181] Rui Huang, Jianming Hu, Yusen Huo, and Xin Pei. Cooperative multi-intersection traffic signal control based on deep reinforcement learning. In *CICTP 2019*, pages 2959–2970. 2019.

[182] Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. Optimizing agent behavior over long time scales by transporting value. *Nature Communications*, 10(1):5223, dec 2019.

[183] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[184] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey 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, apr 2021.

[185] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey 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.

[186] Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. *Advances in neural information processing systems*, 32, 2019.

[187] Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for POMDPs. In *International Conference on Machine Learning*, pages 2117–2126. PMLR, 2018.

[188] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 2961–2970. PMLR, 2019.

[189] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

[190] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, may 2019.

[191] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[192] Vindula Jayawardana, Catherine Tang, Sirui Li, Dajiang Suo, and Cathy Wu. The impact of task underspecification in evaluating deep reinforcement learning. *arXiv preprint arXiv:2210.08607*, 2022.

[193] Ray Jiang, Tom Zahavy, Zhongwen Xu, Adam White, Matteo Hessel, Charles Blundell, and Hado Van Hasselt. Emphatic algorithms for deep reinforcement learning. In *International Conference on Machine Learning*, pages 5023–5033. PMLR, 2021.

[194] Junchen Jin and Xiaoliang Ma. A multi-objective agent-based control approach with application in intelligent traffic signal system. *IEEE Transactions on Intelligent Transportation Systems*, 20(10):3900–3912, 2019.

[195] Wengong Jin, Dr.Regina Barzilay, and Tommi Jaakkola. Hierarchical Generation of Molecular Graphs using Structural Motifs. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4839–4848. PMLR, 2020.

[196] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction Tree Variational Autoencoder for Molecular Graph Generation. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2323–2332. PMLR, 2018.

[197] Jihyuck Jo, Soyoung Cha, Dayoung Rho, and In-Cheol Park. Dsip: A scalable inference accelerator for convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 53(2):605–618, 2017.

[198] Jeff Johns and Sridhar Mahadevan. Sparse approximate policy evaluation using graph-based basis functions. *Dept. Comput. Sci., Univ. Massachusetts Amherst, Amherst, MA, USA, Tech. Rep. UM-CS-2009-041*, 2009.

[199] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International conference on machine learning*, pages 2342–2350. PMLR, 2015.

[200] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[201] Philipp W Keller, Shie Mannor, and Doina Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 449–456, 2006.

[202] Kata Kiatmanaroj, Christian Artigues, Laurent Houssin, and Frédéric Messine. Frequency allocation in a SDMA satellite communication system with beam moving. In *2012 IEEE International Conference on Communications (ICC)*, pages 3265–3269, 2012.

[203] Mirza Golam Kibria, Eva Lagunas, Nicola Maturo, Hayder Al-Hraishawi, and Symeon Chatzinotas. Carrier aggregation in satellite communications: Impact and performance study. *IEEE Open Journal of the Communications Society*, 1:1390–1402, 2020.

[204] Salehin Kibria, Mohammad Tariqul Islam, and Baharuddin Yatim. New compact dual-band circularly polarized universal RFID reader antenna using ramped convergence particle swarm optimization. *IEEE Transactions on Antennas and Propagation*, 62(5):2795–2801, 2014.

[205] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. MOReL : Model-Based Offline Reinforcement Learning. may 2020.

[206] Daeho Kim and Okran Jeong. Cooperative traffic signal control with traffic flow prediction in multi-intersection. *Sensors*, 20(1):137, 2019.

[207] Joanne Taery Kim and Sehoon Ha. Observation Space Matters: Benchmark and Optimization Algorithm. nov 2020.

[208] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Perez. Deep Reinforcement Learning for Autonomous Driving: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–18, 2021.

[209] Randolph Kirchain and Lionel Kimerling. A roadmap for nanophotonics. *Nature Photonics*, 1(6):303–305, 2007.

[210] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2022.

[211] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, and Others. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[212] Louis Kirsch, Sjoerd van Steenkiste, and Juergen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. In *International Conference on Learning Representations*, 2020.

[213] Steven Kisseleff, Bhavani Shankar, Danilo Spano, and Jean-Didier Gayrard. A new optimization tool for mega-constellation design and its application to trunking systems. In *Advances in Communications Satellite Systems. Proceedings of the 37th International Communications Satellite Systems Conference (ICSSC-2019)*, pages 1–15, 2019.

[214] Shunya Kitagawa, Ahmed Moustafa, and Takayuki Ito. Urban traffic control using distributed multi-agent deep reinforcement learning. In *Pacific rim international conference on artificial intelligence*, pages 337–349. Springer, 2019.

[215] A Harry Klopf. *Brain function and adaptive systems: a heterostatic theory.* Number 133. Air Force Cambridge Research Laboratories, Air Force Systems Command, United . . . , 1972.

[216] J Zico Kolter and Andrew Y Ng. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 521–528, 2009.

[217] Ksenia Korovina, Sailun Xu, Kirthevasan Kandasamy, Willie Neiswanger, Barnabas Poczos, Jeff Schneider, and Eric Xing. Chembo: Bayesian optimization of small organic molecules with synthesizable recommendations. In *International Conference on Artificial Intelligence and Statistics*, pages 3393–3403. PMLR, 2020.

[218] John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

[219] Lyudmyla F Kozachenko and Nikolai N Leonenko. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2):9–16, 1987.

[220] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

[221] Srivatsan Krishnan, Behzad Boroujerdian, William Fu, Aleksandra Faust, and Vijay Janapa Reddi. Air Learning: a deep reinforcement learning gym for autonomous aerial robot visual navigation. *Machine Learning*, 110(9):2501–2540, sep 2021.

[222] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. jun 2019.

[223] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[224] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. OpenSpiel: A Framework for Reinforcement Learning in Games. aug 2019.

[225] Greg Landrum et al. Rdkit: A software suite for cheminformatics, computational chemistry, and predictive modeling. *Greg Landrum*, 8, 2013.

[226] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network Randomization: A Simple Technique for Generalization in Deep Reinforcement Learning. oct 2019.

[227] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019.

[228] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. 2020.

[229] Shihui Li, Yi Wu, Xinyue Cui, Honghua Dong, Fei Fang, and Stuart Russell. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4213–4220, 2019.

[230] Yuxi Li. Deep Reinforcement Learning: An Overview. *arXiv preprint arXiv:1701.07274*, 2017.

[231] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-SGD: Learning to Learn Quickly for Few-Shot Learning. jul 2017.

[232] Xianglai Liao, Xin Hu, Zhijun Liu, Shijun Ma, Lexi Xu, Xiuhua Li, Weidong Wang, and Fadhel M. Ghannouchi. Distributed Intelligence: A Verification for Multi-Agent DRL-Based Multibeam Satellite Resource Allocation. *IEEE Communications Letters*, 24(12):2785–2789, dec 2020.

[233] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[234] Cam Linke, Nadia M Ady, Martha White, Thomas Degris, and Adam White. Adapting behavior via intrinsic reward: A survey and empirical study. *Journal of Artificial Intelligence Research*, 69:1287–1332, 2020.

[235] Bo Liu, Ming Ding, Sina Shaham, Wenny Rahayu, Farhad Farokhi, and Zihuai Lin. When machine learning meets privacy: A survey and outlook. *ACM Computing Surveys (CSUR)*, 54(2):1–36, 2021.

[236] Bo Liu, Sridhar Mahadevan, and Ji Liu. Regularized off-policy td-learning. *Advances in Neural Information Processing Systems*, 25, 2012.

[237] De-Rong Liu, Hong-Liang Li, and Ding Wang. Feature selection and feature learning for high-dimensional batch reinforcement learning: A survey. *International Journal of Automation and Computing*, 12(3):229–242, 2015.

[238] Kunpeng Liu, Yanjie Fu, Le Wu, Xiaolin Li, Charu Aggarwal, and Hui Xiong. Automated feature selection: A reinforcement learning perspective. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[239] Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L Gaunt. Constrained Graph Variational Autoencoders for Molecule Design, 2019.

[240] Zhuang Liu, Xuanlin Li, Bingyi Kang, and Trevor Darrell. Regularization Matters in Policy Optimization–An Empirical Study on Continuous Control. *arXiv preprint arXiv:1910.09191*, 2019.

[241] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using

sumo. In *2018 21st international conference on intelligent transportation systems (ITSC)*, pages 2575–2582. IEEE, 2018.

[242] Manuel Loth, Manuel Davy, and Philippe Preux. Sparse temporal difference learning using lasso. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 352–359. IEEE, 2007.

[243] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pages 6379–6390, 2017.

[244] Chris Lu, Jakub Grudzien Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. *arXiv preprint arXiv:2210.05639*, 2022.

[245] Jerry Luo, Cosmin Paduraru, Octavian Voicu, Yuri Chervonyi, Scott Munns, Jerry Li, Crystal Qian, Praneet Dutta, Jared Quincy Davis, Ningjia Wu, et al. Controlling commercial cooling systems using reinforcement learning. *arXiv preprint arXiv:2211.07357*, 2022.

[246] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. Applications of Deep Reinforcement Learning in Communications and Networking: A Survey. *IEEE Communications Surveys & Tutorials*, 21(4):3133–3174, 2019.

[247] Francisco Macedo, M Rosário Oliveira, Antonio Pacheco, and Rui Valadas. Theoretical foundations of forward feature selection methods based on mutual information. *Neurocomputing*, 325:67–89, 2019.

[248] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[249] Amol Mandhane, Anton Zhernov, Maribeth Rauh, Chenjie Gu, Miaosen Wang, Flora Xue, Wendy Shang, Derek Pang, Rene Claus, Ching-Han Chiang, et al. Muzero with self-competition for rate control in vp9 video compression. *arXiv preprint arXiv:2202.06626*, 2022.

[250] Daniel J. Mankowitz, Nir Levine, Rae Jeong, Yuanyuan Shi, Jackie Kay, Abbas Abdolmaleki, Jost Tobias Springenberg, Timothy Mann, Todd Hester, and Martin Riedmiller. Robust Reinforcement Learning for Continuous Control with Model Misspecification. jun 2019.

[251] Daniel J Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, Thomas Köppe, Kevin Millikin, Stephen Gaffney, Sophie Elster, Jackson

Broshear, Chris Gamble, Kieran Milan, Robert Tung, Minjae Hwang, Taylan Cemgil, Mohammadamin Barekatain, Yujia Li, Amol Mandhane, Thomas Hubert, Julian Schrittwieser, Demis Hassabis, Pushmeet Kohli, Martin Riedmiller, Oriol Vinyals, and David Silver. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023.

[252] Timothy A. Mann, Sven Gowal, András György, Ray Jiang, Huiyi Hu, Balaji Lakshminarayanan, and Prav Srinivasan. Learning from Delayed Outcomes via Proxies with Applications to Recommender Systems. jul 2018.

[253] Gérard Maral and Michel Bousquet. *Satellite communications systems: systems, techniques and technology*. John Wiley & Sons, 2011.

[254] James Martens. New insights and perspectives on the natural gradient method. *The Journal of Machine Learning Research*, 21(1):5776–5851, 2020.

[255] Nestor Maslej, Loredana Fattorini, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Helen Ngo, Juan Carlos Niebles, Vanessa Parli, Yoav Shoham, Russell Wald, Jack Clark, and Raymond Perrault. The ai index 2023 annual report. AI Index Steering Committee, Institute for Human-Centered AI, Stanford University, Stanford, CA, 2023.

[256] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement Learning for Combinatorial Optimization: A Survey. mar 2020.

[257] John McCormick. AI-Enabled Cheetos Offer Promise of the Perfect Puff, 2020.

[258] McKinsey & Company. The state of AI in 2020. Technical report, 2020.

[259] McKinsey & Company. Flying across the sea, propelled by AI, 2021.

[260] McKinsey & Company. It's time for businesses to chart a course for reinforcement learning, 2021.

[261] McKinsey & Company. The state of AI in 2022–and a half decade in review, 2023.

[262] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM computing surveys (CSUR)*, 54(6):1–35, 2021.

[263] Hao Mei, Xiaoliang Lei, Longchao Da, Bin Shi, and Hua Wei. Libsignal: An open library for traffic signal control. *arXiv preprint arXiv:2211.10649*, 2022.

[264] David Mendez, Anna Gaulton, A Patrícia Bento, Jon Chambers, Marleen De Veij, Eloy Félix, María Paula Magariños, Juan F Mosquera, Prudence Mutowo, Michał Nowotka, et al. Chembl: towards direct deposition of bioassay data. *Nucleic acids research*, 47(D1):D930–D940, 2019.

[265] Rocío Mercado, Tobias Rastemo, Edvard Lindelöf, Günter Klambauer, Ola Engkvist, Hongming Chen, and Esben Jannik Bjerrum. Graph networks for molecular design. *Machine Learning: Science and Technology*, 2(2):025023, 2021.

[266] Yingjie Miao, Xingyou Song, Daiyi Peng, Summer Yue, John D Co-Reyes, Eugene Brevdo, and Aleksandra Faust. Rl-darts: differentiable architecture search for reinforcement learning. 2021.

[267] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

[268] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*, 2020.

[269] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, jun 2021.

[270] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and Others. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[271] Miltiadis Moralis-Pegios, George Mourgias-Alexandris, Apostolos Tsakyridis, George Giamougiannis, Angelina Totovic, George Dabos, Nikolaos Passalis, Manos Kirtas, T Rutirawut, FY Gardes, et al. Neuromorphic silicon photonics and hardware-aware deep learning for high-speed inference. *Journal of Lightwave Technology*, 40(10):3243–3254, 2022.

[272] Amirhosein Mosavi, Yaser Faghan, Pedram Ghamisi, Puhong Duan, Sina Faizollahzadeh Ardabili, Ely Salwana, and Shahab S. Band. Comprehensive Review of Deep Reinforcement Learning Methods and Applications in Economics. *Mathematics*, 8(10):1640, sep 2020.

[273] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in neural information processing systems*, pages 3303–3313, 2018.

[274] Muddasar Naeem, Syed Tahir Hussain Rizvi, and Antonio Coronato. A Gentle Introduction to Reinforcement Learning and its Application in Different Fields. *IEEE Access*, 8:209320–209344, 2020.

[275] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep Online Learning via Meta-Learning: Continual Adaptation for Model-Based RL. dec 2018.

[276] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. mar 2020.

[277] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications. *IEEE Transactions on Cybernetics*, 50(9):3826–3839, sep 2020.

[278] Mohammad Noaeen, Atharva Naik, Liana Goodman, Jared Crebo, Taimoor Abrar, Zahra Shakeri Hossein Abad, Ana LC Bazzan, and Behrouz Far. Reinforcement learning in urban network traffic signal control: A systematic literature review. *Expert Systems with Applications*, page 116830, 2022.

[279] Northern Sky Research. VSAT and Broadband Satellite Markets. Technical report, 2019.

[280] Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado P van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. *Advances in Neural Information Processing Systems*, 33:1060–1070, 2020.

[281] Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of Cheminformatics*, 9(1):48, dec 2017.

[282] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *International Conference on Machine Learning*, pages 2681–2690. PMLR, 2017.

[283] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving Rubik's Cube with a Robot Hand. oct 2019.

[284] Flor G. Ortiz-Gomez, Daniele Tarchi, Ramón Martínez, Alessandro Vanelli-Coralli, Miguel A. Salas-Natera, and Salvador Landeros-Ayala. Convolutional neural networks for flexible payload management in vhts systems. *IEEE Systems Journal*, 15(3):4675–4686, 2021.

[285] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances in neural information processing systems*, pages 4026–4034, 2016.

[286] Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvari, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado Van Hasselt. Behaviour Suite for Reinforcement Learning. aug 2019.

[287] Blazej Osinski, Adam Jakubowski, Pawel Ziecina, Piotr Milos, Christopher Galias, Silviu Homoceanu, and Henryk Michalewski. Simulation-Based Reinforcement Learning for Real-World Autonomous Driving. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6411–6418. IEEE, may 2020.

[288] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[289] Nils Pachler, Juan Jose Garau Luis, Markus Guerster, Edward Crawley, and Bruce Cameron. Allocating power and bandwidth in multibeam satellite systems using particle swarm optimization. In *2020 IEEE Aerospace Conference*, pages 1–11, 2020.

[290] Nils Pachler, Juan Jose Garau-Luis Luis, Markus Guerster, Edward Crawley, and Bruce Cameron. Allocating Power and Bandwidth in Multibeam Satellite Systems using Particle Swarm Optimization. In *2020 IEEE Aerospace Conference*, pages 1–11. IEEE, mar 2020.

[291] Nils Pachler de la Osa, Markus Guerster, Inigo Portillo Barrios, Edward Crawley, and Bruce Cameron. Static beam placement and frequency plan algorithms for LEO constellations. *International Journal of Satellite Communications and Networking*, page sat.1345, aug 2020.

[292] Christopher Painter-Wakefield and Ronald Parr. Greedy algorithms for sparse reinforcement learning. *arXiv preprint arXiv:1206.6485*, 2012.

[293] Aleix Paris, Inigo del Portillo, Bruce G Cameron, and Edward F Crawley. A Genetic Algorithm for Joint Power and Bandwidth Allocation in Multibeam Satellite Systems. In *2019 IEEE Aerospace Conference*. IEEE, 2019.

[294] Simone Parisi, Simon Ramstedt, and Jan Peters. Goal-driven dimensionality reduction for reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4634–4639. IEEE, 2017.

[295] Unhee Park. A Dynamic Bandwidth Allocation Scheme for a Multi-spot-beam Satellite System. *ETRI Journal*, 34(4):613–616, aug 2012.

[296] Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, Frank Hutter, and Marius Lindauer. Automated reinforcement learning (autorl): A survey and open problems. *Journal of Artificial Intelligence Research*, 74:517–568, 2022.

[297] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 752–759, 2008.

[298] Ronald Parr, Christopher Painter-Wakefield, Lihong Li, and Michael Littman. Analyzing feature generation for value-function approximation. In *Proceedings of the 24th international conference on Machine learning*, pages 737–744, 2007.

[299] Andrew Patterson, Samuel Neumann, Martha White, and Adam White. Empirical design in reinforcement learning. *arXiv preprint arXiv:2304.01315*, 2023.

[300] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. 2021.

[301] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810. IEEE, may 2018.

[302] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[303] A.T.D. Perera and Parameswaran Kamalaruban. Applications of reinforcement learning in energy systems. *Renewable and Sustainable Energy Reviews*, 137:110618, mar 2021.

[304] A.T.D. Perera and Parameswaran Kamalaruban. Applications of reinforcement learning in energy systems. *Renewable and Sustainable Energy Reviews*, 137:110618, 2021.

[305] Athanasios S. Polydoros and Lazaros Nalpantidis. Survey of Model-Based Reinforcement Learning: Applications on Robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, may 2017.

[306] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. Deep reinforcement learning for de novo drug design. *Science Advances*, 4(7):eaap7885, jul 2018.

[307] Niranjani Prasad, Li-Fang Cheng, Corey Chivers, Michael Draugelis, and Barbara E Engelhardt. A Reinforcement Learning Approach to Weaning of Mechanical Ventilation in Intensive Care Units. apr 2017.

[308] Zengyi Qin, Yuxiao Chen, and Chuchu Fan. Density Constrained Reinforcement Learning, 2021.

[309] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[310] Roberta Raileanu and Rob Fergus. Decoupling value and policy for generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 8787–8798. PMLR, 2021.

[311] Roberta Raileanu, Max Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in deep reinforcement learning. *arXiv preprint arXiv:2006.12862*, 2020.

[312] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. mar 2018.

[313] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[314] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized Evolution for Image Classifier Architecture Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4780–4789, jul 2019.

[315] Esteban Real, Chen Liang, David So, and Quoc Le. Automl-zero: Evolving machine learning algorithms from scratch. In *International Conference on Machine Learning*, pages 8007–8019. PMLR, 2020.

[316] Daniele Reda, Tianxin Tao, and Michiel van de Panne. Learning to Locomote: Understanding How Environment Design Matters for Deep Reinforcement Learning. In *Motion, Interaction and Games*, pages 1–10, New York, NY, USA, oct 2020. ACM.

[317] Mali Reda, Fouad Mountassir, and Bousmah Mohamed. Introduction to coordinated deep agents for traffic signal. In *2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, pages 1–6. IEEE, 2019.

[318] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[319] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by Playing - Solving Sparse Reward Tasks from Scratch. feb 2018.

[320] Stefano Giovanni Rizzo, Giovanna Vantini, and Sanjay Chawla. Reinforcement learning with explainability for traffic signal control. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3567–3572. IEEE, 2019.

[321] Stefano Giovanni Rizzo, Giovanna Vantini, and Sanjay Chawla. Time critic policy gradient methods for traffic signal control in complex and congested scenarios. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1654–1664, 2019.

[322] Jan Robine, Tobias Uelwer, and Stefan Harmeling. Discrete Latent Space World Models for Reinforcement Learning. oct 2020.

[323] Brian C Ross. Mutual information between discrete and continuous data sets. *PloS one*, 9(2):e87357, 2014.

[324] Stuart J Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[325] S Salcedo-Sanz and C Bousoño-Calzón. A Hybrid Neural-Genetic Algorithm for the Frequency Assignment Problem in Satellite Communications. *Applied Intelligence*, 22(3):207–217, may 2005.

[326] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. feb 2019.

[327] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, jan 2015.

[328] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[329] Jonathan Schwarz, Jelena Luketina, Wojciech M. Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & Compress: A scalable framework for continual learning. may 2018.

[330] Soheil Mohamad Alizadeh Shabestray and Baher Abdulhai. Multimodal intelligent deep (mind) traffic signal controller. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 4532–4539. IEEE, 2019.

[331] Shitian Shen and Min Chi. Aim low: Correlation-based feature selection for model-based reinforcement learning. *International Educational Data Mining Society*, 2016.

[332] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[333] Lingzhou Shu, Jia Wu, and Ziyan Li. Hierarchical regional control for traffic grid signal optimization. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3547–3552. IEEE, 2019.

[334] Noah Y. Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. Keep Doing What Worked: Behavioral Modelling Priors for Offline Reinforcement Learning. feb 2020.

[335] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, and Others. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[336] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 2017.

[337] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-Driven Behavioral Priors for Reinforcement Learning. nov 2020.

[338] Aleksandrs Slivkins et al. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

[339] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning. may 2019.

[340] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. CURL: Contrastive Unsupervised Representations for Reinforcement Learning. apr 2020.

[341] Niclas Ståhl, Göran Falkman, Alexander Karlsson, Gunnar Mathiason, and Jonas Boström. Deep Reinforcement Learning for Multiparameter Optimization in de novo Drug Design. *Journal of Chemical Information and Modeling*, 59(7):3166–3176, jul 2019.

[342] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[343] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[344] Pei-Hao Su, David Vandyke, Milica Gasic, Nikola Mrksic, Tsung-Hsien Wen, and Steve Young. Reward Shaping with Recurrent Neural Networks for Speeding up On-Line Policy Learning in Spoken Dialogue Systems. aug 2015.

[345] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Flores Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, and Others. Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward. In *AAMAS*, pages 2085–2087, 2018.

[346] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H S Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[347] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

[348] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[349] István Szita. Reinforcement learning in games. *Reinforcement Learning: State-of-the-art*, pages 539–577, 2012.

[350] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE, sep 2017.

[351] Victor Talpaert, Ibrahim Sobh, B Ravi Kiran, Patrick Mannion, Senthil Yogamani, Ahmad El-Sallab, and Patrick Perez. Exploring applications of deep reinforcement learning for real-world autonomous driving systems. jan 2019.

[352] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PLOS ONE*, 12(4):e0172395, apr 2017.

[353] Youhai Tan, Lingxue Dai, Weifeng Huang, Yinfeng Guo, Shuangjia Zheng, Jinping Lei, Hongming Chen, and Yuedong Yang. Drlinker: Deep reinforcement learning for optimization in fragment linking design. *Journal of Chemical Information and Modeling*, 62(23):5907–5917, 2022.

[354] Voot Tangkaratt, Jun Morimoto, and Masashi Sugiyama. Model-based reinforcement learning with dimension reduction. *Neural Networks*, 84:1–16, 2016.

[355] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite. jan 2018.

[356] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward Constrained Policy Optimization. may 2018.

[357] Edward Lee Thorndike. *Animal intelligence: Experimental studies*. Transaction Publishers, 1911.

[358] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive Multiview Coding. jun 2019.

[359] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, sep 2017.

[360] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[361] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, oct 2012.

[362] Yao-Hung Hubert Tsai, Paul Pu Liang, Amir Zadeh, Louis-Philippe Morency, and Ruslan Salakhutdinov. Learning Factorized Multimodal Representations. jun 2018.

[363] Mikhail L'vovich Tsetlin et al. *Automaton theory and modeling of biological systems*, volume 102. Academic Press New York, 1973.

[364] Jorge R Vergara and Pablo A Estévez. A review of feature selection methods based on mutual information. *Neural computing and applications*, 24:175–186, 2014.

[365] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR, 2017.

[366] Florian Vidal, Hervé Legay, George Goussetis, Maria Garcia Vigueras, Ségolène Tubau, and Jean-Didier Gayrard. A methodology to benchmark flexible payload architectures in a megaconstellation use case. *International Journal of Satellite Communications and Networking*, 39(1):29–46, jan 2021.

[367] Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Munchausen reinforcement learning. *Advances in Neural Information Processing Systems*, 33:4235–4246, 2020.

[368] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, and Others. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

[369] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[370] Marin Vlastelica, Michal Rolínek, and Georg Martius. Neuro-algorithmic policies enable fast combinatorial generalization. *arXiv preprint arXiv:2102.07456*, 2021.

[371] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M Lucas, Adam Smith, and Sebastian Risi. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the genetic and evolutionary computation conference*, pages 221–228, 2018.

[372] Kiri Wagstaff. Machine Learning that Matters. jun 2012.

[373] Jane X. Wang, Michael King, Nicolas Porcel, Zeb Kurth-Nelson, Tina Zhu, Charlie Deck, Peter Choy, Mary Cassin, Malcolm Reynolds, Francis Song, Gavin Buttimore, David P. Reichert, Neil Rabinowitz, Loic Matthey, Demis Hassabis, Alexander Lerchner, and Matthew Botvinick. Alchemy: A structured task distribution for meta-reinforcement learning. feb 2021.

[374] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Poet: open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 142–151, 2019.

[375] Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeff Clune, and Kenneth O. Stanley. Enhanced POET: Open-Ended Reinforcement Learning through Unbounded Invention of Learning Challenges and their Solutions. mar 2020.

[376] Yu-Xiang Wang, Alekh Agarwal, and Miroslav Dud\ik. Optimal and adaptive off-policy evaluation in contextual bandits. In *International Conference on Machine Learning*, pages 3589–3597. PMLR, 2017.

[377] David Warde-Farley, Tom Van de Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih. Unsupervised Control Through Non-Parametric Discriminative Rewards. nov 2018.

[378] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[379] Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z. Leibo, Adam Santoro, Mevlana Gemici, Malcolm Reynolds, Tim Harley, Josh Abramson, Shakir Mohamed, Danilo Rezende, David Saxton, Adam Cain, Chloe Hillier, David Silver, Koray Kavukcuoglu, Matt Botvinick, Demis Hassabis, and Timothy Lillicrap. Unsupervised Predictive Memory in a Goal-Directed Agent. mar 2018.

[380] Hua Wei, Chacha Chen, Guanjie Zheng, Kan Wu, Vikash Gayah, Kai Xu, and Zhenhui Li. Presslight: Learning max pressure control to coordinate traffic signals in arterial network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1290–1298, 2019.

[381] Hua Wei, Nan Xu, Huichu Zhang, Guanjie Zheng, Xinshi Zang, Chacha Chen, Weinan Zhang, Yanmin Zhu, Kai Xu, and Zhenhui Li. Colight: Learning network-level cooperation for traffic signal control. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1913–1922, 2019.

[382] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. Recent advances in reinforcement learning for traffic signal control: A survey of models and evaluation. *ACM SIGKDD Explorations Newsletter*, 22(2):12–18, 2021.

[383] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

[384] Robin Winter, Floriane Montanari, Frank Noé, and Djork-Arné Clevert. Learning continuous and data-driven molecular descriptors by translating equivalent chemical representations. *Chemical Science*, 10(6):1692–1701, 2019.

[385] Ian H Witten. An adaptive optimal controller for discrete-time markov environments. *Information and control*, 34(4):286–295, 1977.

[386] Robert S Woodworth and Harold Schlosberg. Experimental psychology, rev. 1954.

[387] Cathy Wu, Aboudy Kreidieh, Eugene Vinitsky, and Alexandre M Bayen. Emergent behaviors in mixed-autonomy traffic. In *Conference on Robot Learning*, pages 398–407. PMLR, 2017.

[388] Yifan Wu, George Tucker, and Ofir Nachum. Behavior Regularized Offline Reinforcement Learning. nov 2019.

[389] Zhirong Wu, Yuanjun Xiong, Stella Yu, and Dahua Lin. Unsupervised Feature Learning via Non-Parametric Instance-level Discrimination. may 2018.

[390] Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael D. Thomure, Houmehr Aghabozorgi, Leon Barrett, Rory Douglas, Dion Whitehead, Peter Dürr, Peter Stone, Michael Spranger, and Hiroaki Kitano. Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, feb 2022.

[391] Annie Xie, James Harrison, and Chelsea Finn. Deep Reinforcement Learning amidst Lifelong Non-Stationarity. jun 2020.

[392] Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. Prediction-Guided Multi-Objective Reinforcement Learning for Continuous Robot Control. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[393] Mengdi Xu, Zuxin Liu, Peide Huang, Wenhao Ding, Zhepeng Cen, Bo Li, and Ding Zhao. Trustworthy reinforcement learning against intrinsic vulnerabilities: Robustness, safety, and generalizability. *arXiv preprint arXiv:2209.08025*, 2022.

[394] Zhongwen Xu, Hado P van Hasselt, Matteo Hessel, Junhyuk Oh, Satinder Singh, and David Silver. Meta-gradient reinforcement learning with an objective discovered online. *Advances in Neural Information Processing Systems*, 33:15254–15264, 2020.

[395] Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[396] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. *Advances in Neural Information Processing Systems*, 32, 2019.

[397] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6410–6421. Curran Associates, Inc., 2018.

[398] Chao Yu, Jiming Liu, and Shamim Nemati. Reinforcement Learning in Healthcare: A Survey. aug 2019.

[399] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient Surgery for Multi-Task Learning. jan 2020.

[400] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100. PMLR, 2020.

[401] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: Model-based Offline Policy Optimization. may 2020.

[402] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3562–3573, 2018.

[403] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning Invariant Representations for Reinforcement Learning without Reconstruction. jun 2020.

[404] Baohe Zhang, Raghu Rajan, Luis Pineda, Nathan Lambert, André Biedenkapp, Kurtland Chua, Frank Hutter, and Roberto Calandra. On the importance of hyperparameter optimization for model-based reinforcement learning. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 4015–4023. PMLR, 13–15 Apr 2021.

[405] Baohe Zhang, Raghu Rajan, Luis Pineda, Nathan Lambert, André Biedenkapp, Kurtland Chua, Frank Hutter, and Roberto Calandra. On the importance of hyperparameter optimization for model-based reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 4015–4023. PMLR, 2021.

[406] Daniel Zhang, Nestor Maslej, Erik Brynjolfsson, John Etchemendy, Terah Lyons, James Manyika, Helen Ngo, Juan Carlos Niebles, Michael Sellitto, Ellie Sakhaee, Yoav Shoham, Jack Clark, and Raymond Perrault. The AI Index 2022 Annual Report. Technical report, AI Index Steering Committee, Stanford Institute for Human-Centered AI, Stanford University, 2022.

[407] Huichu Zhang, Siyuan Feng, Chang Liu, Yaoyao Ding, Yichen Zhu, Zihan Zhou, Weinan Zhang, Yong Yu, Haiming Jin, and Zhenhui Li. Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In *The world wide web conference*, pages 3620–3624, 2019.

[408] Pei Zhang, Xiaohui Wang, Zhiguo Ma, Shuaijun Liu, and Junde Song. An online power allocation algorithm based on deep reinforcement learning in multibeam satellite systems. *International Journal of Satellite Communications and Networking*, 38(5):450–461, sep 2020.

[409] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep Learning Based Recommender System. *ACM Computing Surveys*, 52(1):1–38, feb 2019.

[410] Chenyang Zhao, Olivier Sigaud, Freek Stulp, and Timothy M Hospedales. Investigating generalisation in continuous deep reinforcement learning. *arXiv preprint arXiv:1902.07015*, 2019.

[411] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE, 2020.

[412] Alex Zhavoronkov, Yan A. Ivanenkov, Alex Aliper, Mark S. Veselov, Vladimir A. Aladinskiy, Anastasiya V. Aladinskaya, Victor A. Terentiev, Daniil A. Polykovskiy, Maksim D. Kuznetsov, Arip Asadulaev, Yury Volkov, Artem Zholus, Rim R. Shayakhmetov, Alexander Zhebrak, Lidiya I. Minaeva, Bogdan A. Zagribelnyy, Lennart H. Lee, Richard Soll, David Madge, Li Xing, Tao Guo, and Alán Aspuru-Guzik. Deep learning enables rapid identification of potent DDR1 kinase inhibitors. *Nature Biotechnology*, 37(9):1038–1040, sep 2019.

[413] Fei Zheng, Zhao Pi, Zou Zhou, and Kaixuan Wang. LEO Satellite Channel Allocation Scheme Based on Reinforcement Learning. *Mobile Information Systems*, 2020:1–10, dec 2020.

[414] Fei Zheng, Zhao Pi, Zou Zhou, and Kaixuan Wang. Leo satellite channel allocation scheme based on reinforcement learning. *Mobile Information Systems*, 2020:8868888, 2020.

[415] Guanjie Zheng, Yuanhao Xiong, Xinshi Zang, Jie Feng, Hua Wei, Huichu Zhang, Yong Li, Kai Xu, and Zhenhui Li. Learning phase competition for traffic signal control. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1963–1972, 2019.

[416] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph Neural Networks: A Review of Methods and Applications. dec 2018.

[417] Pengyuan Zhou, Tristan Braud, Ahmad Alhilal, Pan Hui, and Jussi Kangasharju. Erl: Edge based reinforcement learning for optimized urban traffic light control. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 849–854. IEEE, 2019.

[418] Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N. Zare, and Patrick Riley. Optimization of Molecules via Deep Reinforcement Learning. *Scientific Reports*, 9(1):10752, dec 2019.

[419] Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real world robotic reinforcement learning. In *International Conference on Learning Representations*, 2020.

[420] Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The Ingredients of Real-World Robotic Reinforcement Learning. *arXiv preprint arXiv:2004.12570*, 2020.

[421] Zhaocheng Zhu, Chence Shi, Zuobai Zhang, Shengchao Liu, Minghao Xu, Xinyu Yuan, Yangtian Zhang, Junkun Chen, Huiyu Cai, Jiarui Lu, et al. Torchdrug: A powerful and flexible machine learning platform for drug discovery. *arXiv preprint arXiv:2202.08320*, 2022.

[422] Zhuangdi Zhu, Kaixiang Lin, Anil K Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

[423] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.