The Credit Assignment Problem in Reinforcement Learning

A dissertation presented by

Eduardo Pignatelli

Submitted to **UCL** in partial fulfilment

of the requirements for the degree of

Philosophiae Doctor

Supervisor:

Laura Toni

Subsidiary supervisor:

Tim Rocktäschel

Department of Electronic and Electrical Engineering

University College London

September 2025

I, Eduardo Pignatelli, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Of all men's miseries the bitterest is this: to know so much and to have control over nothing.

Herotodus, Histories.

Abstract

Reinforcement Learning (RL) has made significant progress in a variety of domains, from playing games to controlling robots. However, while RL works reasonably well in problems when rewards are dense and immediate, it becomes significantly harder when these are *sparse* and *delayed*. This is the case of most real-world decision problems: they take a *long time to complete*, and they seldom provide immediate feedback, but often *with delay* and little insight as to which actions caused it.

The problem of learning to associate actions with their *long-term*, outcomes is known as the temporal *Credit Assignment Problem (CAP)*: *to distribute the credit of success among the multitude of decisions involved* (Minsky, 1961).

This dissertation stems from the idea that improving the ability to *predict* – i.e., to *assign credit* – is the most effective way to enhance the agents' ability to make optimal decisions – to *control* – in a broad range of tasks. The manuscript is then a collection of experiments and theoretical contributions with two aims: to better understand the CAP, and to propose new methods to address it.

We provide a comprehensive survey of the field, the first after the CAP was first introduced by Minsky (1961). We realign the original CAP to Deep RL, organise the set of methods into a coherent perspective, and define a call for action for scaling RL to real-world problems. On this call, we then investigate AI-assisted RL, using the prior knowledge and reasoning capabilities of Large Language Models (LLMs) to assist and supervise RL training. Finally, we focus on closing the gap between the infeasible computational demand of RL and the limited resources available in academia, reimplementing MiniGrid, a popular RL benchmark, in a more efficient and scalable way.

Impact

This dissertation aims make a small step towards the development of more efficient and scalable RL algorithms, with the ultimate goal of enabling the deployment of RL in real-world, complex problems.

The work is motivated by the observation that RL works well in problems with dense and immediate rewards, but it struggles in problems with sparse and delayed rewards. This is, in fact, the case of most real-world decision problems, which are usually composed of a large number of sequential, often hierarchical, decisions, and where the feedback is often but only at the end of the task.

To contribute to this challenge, the manuscript first reviews the current state of the field, providing a comprehensive survey of the literature on the CAP in RL: What is the CAP that we are actually trying to solve? Are we all solving the same problem, or is it unambiguously defined? What are the main methods that have been proposed to address it? What are the main challenges that we face when trying to scale RL to real-world problems?

Having established and formalised the problem, it then identifies a major obstacle to the scaling of RL to real-world problems: human decision-making often works at a higher level of abstraction than the low-level actions that RL algorithms are trained on. This gap is usually met with the introduction of temporally extended actions (options), which provide a way to search into an action (option) space that is more aligned with the human decision-making process. However, these are often hard to learn, must be specified by humans *ex-ante* and require a large amount of computational resources to train.

To address this, the manuscript proposes a new approach to the problem, where the prior knowledge and reasoning capabilities of LLMs are used to assist and supervise the training of RL algorithms. This introduces a promising direction for future RL training, where the agent can learn in a higher-level action space, and where the human intervention is reduced to a minimum.

Finally, a key impact is in the evaluation protocol of RL algorithms, which often require a large amount of computational resources to train. In its last chapters, the dissertation analyses the computational bottlenecks of RL algorithms, and identifies the interface with the environment as a major bottleneck. To address this, it proposes NAVIX, a new, more efficient and scalable implementation of MiniGrid, a popular RL benchmark. Thanks to a substantial increase of throughput, NAVIX obtains over 200 000× speedup compared to the original MiniGrid implementation. This speedup, allows not only to train RL algorithms faster, but it opens a whole new paradigm of research for RL agents.

Acknowledgements

The brightest thank you is to my supervisor, Laura Toni, who has been a constant source of inspiration, guidance, and unconditional support throughout this journey. Her candid concession to freely investigate without boundaries, and, at the same time, the constant push to challenge my ideas, my believes and the result of my experiments have been an invaluable resource to succeed and improve. I am grateful for the trust she has placed in me, and for the freedom she has given me to decide my path autonomously, a higher risk, but hopefully a higher reward journey. I am truly thankful to my second supervisor, Tim Rocktäschel, who has been an invaluable mentor, and showed me his way of thinking, his way of approaching problems, and his way of conducting research. Every time I have had the need for guidance, advice, constructive and insightful critique, and an eye with a different, experienced perspective, he has been there despite his interminably busy schedule.

Among my collaborators, I am especially thankful to Johan Ferret, who I have been harassing for the whole length of my PhD with questions, ideas, and requests, to which he has always responded with patience, kindness, and a constructive and insightful attitude. I am very grateful to all my co-authors, who I have harassed for a shorter period of time, but with the same intensity: Hado van Hasselt, Thomas Mesnard, Matthieu Geist, Olivier Pietquin, Edward Grefenstette, Davide Paglieri, Samuel Coward, Aya Kayal, Robert Tjarko Lange, Pablo Samuel Castro, Jarek Liesen and Chris Lu. My research would not have been what it is today without their contributions.

Thank you to my MPhil examiner Ilija Bogunovic for his feedback and his guidance that helped me kickstart the investigations. Thank you to Anil Bharath, for its unconditional support to my research.

I am grateful to all my beautiful colleagues and friends of the LASP, Alan Guedes, Nagham Osman, Pedro Gomes, Aya Kayal, Edoardo Gruppi, Keyue Jiang, Han Wang, Silvia Rossi, and Sephora Madjiheurem, for the countless hours of discussions, debates, and second opinions. Thank you for always making me feel at home, constantly making my day brighter, and being my family at UCL.

Thank you to my colleagues at UCL, who have been a constant source of motivation to showing up in the office every day: Fan Yang, Vittorio Casagrande, Shyam Ramesh, Amany Kassem, Afroditi Papadaki, Gianluca Anselmi, Navin Keizer, Iman Valiulahi, Martin Ferianc, Fabio Palmese, and many others.

Thank you to all the members of the UCL AI Centre, for the discussions, the debates, second opinions, and the constant source of inspiration.

Thank you to Al Fisher, for having been my mentor before the PhD, for the understanding the hard moments with empathy and professionalism, and for showing me his truly radically candid approach to management. It has been an invaluable lesson that I will carry with me for the rest of my life.

I would not have been where I am today without the support, and the love of my wife, who constantly and consistently seconded my madness, helped me reflect and analyse my choices and unwind when I needed it the most. I would not have been where I am today without the support and the affection of my father, my mother, their partners and my brother who have always been there for me, and who have always believed in me.

A special thank you to my family somewhere around the world, Gabriele Mirra, Antonio Stanziola and Sara Russo, Gianpiero Lambiase and Sabrina Valentino for being my partners in crime through the years, for the fun, for their indispensable contributions to my mental health, and for always being there for me.

Thank you to my friends in or from Naples, Enzo Barbato, Valerio Abbruzzo, Mirko D'Angiò, Paolo and Diego Frolli, Alessandra Guglielmo, Fabrizio Ciardulli, Luca Benedini, Priya Guha, who have always been a cosy place to come back to, or made a new place feel like home. Thank you for your advice and for reminding me of the importance of the simple things in life.

Thank you to all the other people that have been part of my journey, and that I did not have the chance to mention here, and that have contributed to my growth and my development.

Thank you to the EPSRC, who has funded this PhD. I am grateful for the financial support that has allowed me to focus on my research.

Contents

List of Figures

List of Tables

"A computer program [...] can be seen simultaneously as a set of computer instructions or as concrete poetry formed by the indentations in the text of the program."

Nicholas Negroponte. Being Digital, 1995.

Part I

Contexts

Chapter 1

Introduction

1.1 A science of decision-making

There is abundant literature trying to understand how humans make decisions, and its corpus is inherently **multidisciplinary**. Multiple domains explore the topic from various angles, such as economics, psychology, neuroscience, or philosophy, resulting in a palimpsest that transcends them all (Buchanan, 2005). Psychologists observe associations between individual choices and their contextual needs (Thorndike, 1898; Pavlov, 1927; Skinner, 1953). Neuroscientists analyse the internal phenomena that allow for the acquisition, organisation, processing, and retrieval of information (Shiv et al., 2005; Fellows, 2004; Vartanian & Mandel, 2011). Economists resort to discounted utility theory and mechanism design to explain how preferences are formed and how these generate rational choices (Samuelson, 1937; Edwards, 1954; Simon, 1959b). Western philosophers ponder the assumptions under decision-making, questioning how to reconcile the naturalism of physics with the intentionality of consciousness (Deacon, 2012). The field is vast, and the literature is abundant, but the core question remains the same: *How do we make decisions?*

The complexity of the question constantly stimulated cross-disciplinary interactions, and, when cross-fertilisation has been prolific, a new theory merged two subfields into a whole new one, such as behavioural economics (Simon, 1957; 1959b), neuropsychology (Klopf, 1972), or cognitive psychology (Piaget, 1962).

Today, this *interdisciplinary* apparatus is not only a way to solve problems that are too complex for a single discipline, but also a way to better *specify* the problem itself, which, as we will see later in the manuscript, is not always unambiguously defined. These disciplines benefit from mutual confirmation: psychology resorts to evidence from neurobiology to consolidate their results; economists motivate micro-economic decisions on psychologist's questions and experimental methods; philosophers start from the assumptions of physics to integrate meaning into reality. In short, to understand how agencies make decisions, we cannot limit the investigation to a single field, and to find answers it is important to analyse the problem from multiple angles.

Upon this multidisciplinary backdrop lay the foundations of decision-making in machines, and the science of the *ifs* and *hows* machines can *learn* to make decisions. The field of **Artificial Intelligence (AI)** – the art of computers acquiring the ability to make decisions similar and eventually better to human intelligence – has for long harvested from this interdisciplinary compound of knowledge. RL has arisen as a contribution to the debate, and now blends elements from control theory (Bellman, 1957), such as the Bellman equations; utility theory (Sutton, 1984) and economics (Samuelson, 1937) to quantify secondary rewards; neuroscience (Quartz et al., 1993; Schultz et al., 1997; Glimcher, 2011) to explain, interpret and design reward-based algorithms; physiology (Barto, 1995; Houk & Adams, 1995) and psychology (Kamin, 1969; Recorla & Wagner, 1972; Sutton & Barto, 1981) to induce useful biases in computer algorithms.

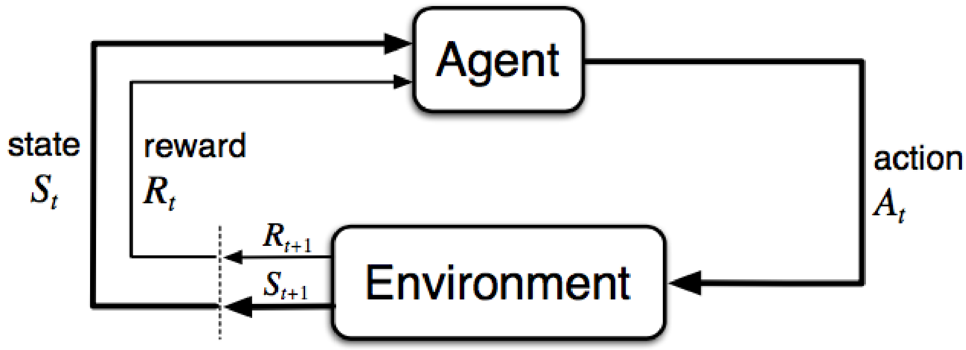


Figure 1.1: The modern formulation of a reinforcement learning problem as summarised by Sutton (Sutton & Barto, 2018). The agent observes the environment and takes suitable action, which, in turn, modifies the environment itself. After the interaction, the agent reads the reward signal, which the environment emits, providing him with information on the directions of actions. This exchange is sequential and recursive, and can be infinite or finite, respectively in continuing and episodic settings.

1.2 Reinforcement Learning

Reinforcement Learning (Sutton & Barto, 2018) formalises the problem of animal learning in mathematical and computational terms, as shown in Figure 1.1. A program (the *agent*) interacts with its surroundings (the *environment*) by making a decision (the *action*). The action is the agent's interface with the environment. Before each action, the agency may use its sensory apparatus to *observe* the environment and take suitable decisions. After each action, the agent may read a feedback signal from the environment (the *reward*), whose state it just modified. The goal of the agent is to learn a rule of behaviour (*the policy*) that maximises a sum of rewards.

Let us develop a better intuition with an example. Every dog owner knows how to reinforce aspects of their pet's behaviour with rewards. For instance, imagine that we present the dog with a button, and every time it pushes the button, we reward our friend with a tasty snack. After repeated interactions, the dog learns to press the button when it is hungry, associating and *anticipating* the receipt of food. In this example, the dog is the *agent*, acting in the surroundings of its environment. The behaviour "pressing the button" is the *operant*; the act of collecting food, the *reinforcer*; and the rule by which it presses the button to receive the food when it perceives hunger is the *policy*. Notice that these are not *reflexes*, but cognitive responses resulting from the ability of the dog to *predict* the effects of its action, and the awareness to modify its own behaviour to receive a prize.

Today, the field of RL is a vibrant and active area of research. RL is poised to impact many real-world problems that require specialistic sequential decision making, such as strategy (Silver et al., 2016; 2018; Schrittwieser et al., 2020; Anthony et al., 2020; Vinyals et al., 2019; Perolat et al., 2022) and arcade video games (Mnih et al., 2013; 2015; Badia et al., 2020; Wurman et al., 2022), climate control (Wang & Hong, 2020), energy management (Gao, 2014), car driving (Filos et al., 2020) and stratospheric balloon navigation (Bellemare et al., 2020), designing circuits (Mirhoseini et al., 2020), cybersecurity (Nguyen & Reddi, 2021), robotics (Kormushev et al., 2013), or physics (Degrave et al., 2022), aligning LLMs to human preferences (Christiano et al., 2017; Brown et al., 2020; Ouyang et al., 2022).

1.3 The Credit Assignment Problem

One fundamental mechanism allowing RL agents to succeed in the scenarios above is their ability to *predict* the *effects* of their actions over outcomes – e.g., a win, a loss, a particular event, a payoff. Often, these outcomes are consequences of isolated decisions taken in a very remote past: actions can have long-term effects. The problem of learning to associate actions with distant, future outcomes is known as the temporal Credit Assignment Problem (CAP): *to distribute the credit of success among the multitude of decisions involved* (Minsky, 1961). Overall, the *influence* that an action has on the achievement of an outcome represents *knowledge* in the form of *associations* between world states, actions and outcomes (Sutton et al., 2011; Zhang et al., 2020a). These associations constitute the scaffolding that agencies can use to deduce, reason, improve and act to address decision-making problems.

To visualise the CAP, let us re-consider the dog example. To exacerbate the problem, let us assume that the tasty snack is not delivered immediately after the button is pressed, but the day after. The dog must now learn to associate button presses with receiving a snack, even if the action responsible for the reward is *delayed* and separated from it by a long sequence of irrelevant events and other actions, like staying quiet, going outside, or chasing a squirrel. This is the essence of the CAP: to learn to associate actions with outcomes separated by a long sequence of irrelevant events.

Significance. While RL works reasonably well when rewards are dense and immediate, it becomes significantly harder when these are **sparse** and **delayed**. Yet, this is the case of most real-world decision problems: they take a *long time to complete*, and they seldom provide immediate feedback, but often *with delay* and little insight as to which actions caused it. Since the purpose of the CAP is exactly that to deal with delayed rewards, it becomes clear that advancing the CAP is paramount to solve RL: to build an AI able to solve a broad variety of real-world tasks safely and reliably.

These conditions materialise in RL with *task completion* (Efroni et al., 2021) settings, where the agent receives a reward signal only at termination. Here the learning signal is weak, noisy, or deceiving, and the ability to separate serendipitous outcomes from those caused by informed decision-making becomes a hard challenge. In such scenarios, rewarding events are rare, and Deep RL agents often struggle to convert occasional successes due to random exploration[1] into a robust decision-making process. Furthermore, as these environments grow in complexity with the aim to scale to real-world tasks (Rahmandad et al., 2009; Luoma et al., 2017), the actions taken by an agent affect an increasingly vanishing part of the outcome, and it becomes even harder to accurately identify

[1] or, rather, according to their exploration rule.

the causes of success or failure.

As a result, tasks that are usually easy to solve for humans become hard to address for RL agents, and there is a large gap in the literature between the two. This dissertation aims to work towards bridging this gap, from a better understanding of the problem to proposing new methods to address it, and to evaluate it.

1.4 A case for model-free RL

RL scholars agree on two major avenues to evaluate actions of RL agents, advance the field and bring them to equate and eventually surpass human performance in everyday tasks:

(*a*) **Model-based** RL aims to learn a model of the environment by predicting its next state, after the agent has taken a certain action. This usually leads to choosing the best action by *planning* in a goal-oriented way: to simulate the outcome of as many decisions as possible up to a remote future and choose the one yielding the highest sum of rewards.

(*b*) **Model-free** RL, instead, aims to model only a partial future, one specific statistic of it: the sum of all rewards. This usually leads to value-based RL, which, instead, chooses the best actions based on predictions alone. Values simulate a form of habitual behaviour: the agent *knows immediately*[2] which is the best decision, without simulating the final, future state.

In the next paragraphs, we discuss the dichotomy between model-based and model-free methods, and briefly summarise the arguments for employing one or the other.

[2]We may say instinctively, were we talking about humans.

Model-based RL fundamentally aims to model the evolution of the state of the world, given *(a)* its governing laws, and *(b)* the agent's decisions. To model this exactly, one requires modelling information that may not be useful to the agent's decision-making process, since not all the information in the state contributes to decision-making. For example, imagine having to choose the dessert at a restaurant. Whether outside is raining or not will not (for the most of us) have any influence on our decision. We may focus on other characteristics, such as how hungry we are, how much we are craving for a sweet treat, or prior personal preferences. Despite that, if we were to follow a model-based approach, a model of the world would also include whether it is raining or not, resulting in a potentially waste of knowledge capacity and computation resources for something that does not impact the agent's decision making.

On the contrary, a predictive model of the reward represents the **bare minimum** information for optimal decisions (Silver et al., 2021). Indeed, were we to model only the reward, free from any other information about world-state transitions, we could avoid paying the price of modelling useless information, and focus only on the features of the world that affect our decision making. This is the case of model-free RL, at whose core is the CAP.

One argument against model-free RL is that this useless information – for example, the weather in the previous example – is not useful *right now*, but it could be later, should the agent decide to pursue a new, different goal. This is a valid argument if we had a perfect model of the environment readily available, which is rarely the case in RL problems, and especially in the real world: world models are often learnt from experience themselves (Kaiser et al., 2020). In these conditions, modelling also all the non-useful information, with the hope that they will be useful someday, somehow, is a problem too large to tackle.

Finally, to further argue for model-free methods, evidence from neuropsychology and philosophical studies suggest that humans do not represent the world as it is, and this biased representation inherently expresses *preferences* over something that is possible in the future (Hoffman, 2016; Prakash et al., 2021; Le Lan et al., 2022). In other words, it is suboptimal to represent the world exactly as it evolves, and because natural agents evolved this way, this is likely beneficial also to artificial agents.

As with many other problems, it is very likely that *virtus media vox*, and that a mixture of model-based and model-free RL is the most effective way to scale decision making to the real world. In this dissertation, we assume that RL requires both approaches, but we focus on **model-free methods**, at whose core is the CAP.

1.5 Frontiers

Having established our focus on model-free approaches, and their significance in scaling RL to the real world, in this section, we discuss the challenges that the field of RL faces today, and how the CAP is at the core of these challenges. This provides a clear picture of the research frontiers, and sets the stage for the research questions that we aim to address in this dissertation. This list is not exhaustive, but it is representative of the current state of the field, and informative to where to direct future research efforts. We identify five, large categories of challenges.

Challenge 1 (The problem is ambiguously defined.). First, the literature offers no formal definition of *credit*, *credit assignment*, and *credit assignment method*. Many works claim to work on the CAP, but it is yet not clear what mathematical problem these works are referring to. *What is credit? What does it mean to assign credit? How can we quantify the influence of an action on an outcome? What is the mathematical form of the CAP? What do these terms mean mathematically? What does "assigning credit" mean formally? Are we all solving the same problem, or is the science misled by multiple undercover* *problems with the same name?*

Challenge 2 (Temporal CA is inefficient.). Second, current methods still struggle to assign credit for long sequences. State-of-the-art algorithms in RL, such as Proximal Policy Optimisation (PPO) (Schulman et al., 2017), Deep Q-Network (DQN) (Mnih et al., 2015), or Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016), rely on Temporal Difference (TD) errors, or, more in general, to use the temporal vicinity between the action and the reward to quantify the contribution of an action. In our one-day delayed snack, dog example, these methods would assign large credit to the actions taken just before the reward, including waking up, roaming around. By experiment construction, these are irrelevant – even potentially detrimental – to receiving the reward. On the contrary, since the action *press the button* had been taken at the very beginning of the sequence, separated by a full day from the reward, canonical RL methods would assign very small credit to the action, even if that is the one action that caused the reward. Overall, this causes the agent to learn slowly, and to be sample-inefficient (Ye et al., 2021; Kapturowski et al., 2023); solutions are often brittle to the hyperparameter choice (Henderson et al., 2018), inelastic to generalise zero-shot to different tasks (Kirk et al., 2023), and prone to overfitting (Behzadan & Hsu, 2019; Wang et al., 2022).

Challenge 3 (Existing methods do not scale.). Third, there exist CA methods that are very promising, but that do not scale to complex problems, either because they are computationally expensive, or because they require human interventions. For example, some frameworks, such as options (Sutton et al., 1999) and reward shaping (Ng et al., 1999), offer great promises to advance the CA problem. These methods rely on human labels. A shaping function or an option should be carefully designed to provide useful guidance without leading to unintended behaviours. This often calls for incorporating domain knowledge or heuristics about the task. Such knowledge may not be readily available or easily codifiable, limiting the applicability of reward shaping in diverse or unknown environments. This process is complex and time-consuming, and it might not always be possible to devise a reward function that incentivizes learning, is computationally cheap, and general enough to adapt to various tasks. A set of new methods (Buesing et al., 2019; Mesnard et al., 2021; 2023) is promising, but has not yet been tested at scale, in particular to understand their limitations and advantages in building a foundation model for RL.

Challenge 4 (RL lacks a foundation model.). Fourth, methods often learn *tabula rasa*, i.e., from scratch. This is a significant drawback, as RL agents typically begin with no prior knowledge, and it requires them to learn the nuances and intricacies of complex tasks from scratch, often hindering the learning process. In these settings, the lack of controlled experimental conditions, such as the ability to observe counterfactuals, makes it difficult for them to distinguish between correlation and causation, and without a prior understanding of the task, learning is very sample inefficient. A foundation model for RL – that is, a general, robust and efficient **value function** for a broad range of observations and tasks – is a fundamental, and yet missing element in the puzzle to scale RL to the real world.

Challenge 5 (Evaluations are inconclusive.). Fifth, the literature offers a set of interesting benchmarks, each with its own subset of perks, but it overall lacks a precise protocol to combine them for an appropriate evaluation (Agarwal et al., 2021). Are efforts well coordinated to provide a comprehensive view of the problem? Or are methods tested on a subset of tasks that do not reflect the complexity of the problem? Are the benchmarks used to evaluate the methods representative of the real-world challenges?

1.5.1 Explicit research questions

With these challenges in mind, we start our investigation. This dissertation sets out to investigate how the ability to *predict* – i.e., to *assign credit* – affects the agents' ability to make optimal decisions – to *control*. The manuscript is a collection of studies and experiments to better *understand* the CAP, to *address* it, and to *evaluate* the proposed solutions.

Formally, the investigation attends to the following three questions.

Question 1. *What is credit? What does it mean to assign credit? How can we quantify the influence of an action on an outcome? What is the mathematical form of the CAP? Are methods all solving the same problem, or is researched misled by multiple problems under the same name?*

Answering Question 1 helps to shed light on Challenges 1 and 2. In Chapters 4-8 we propose a unifying formalism for the CAP that takes into account the most recent methods to assign credit. This enables a fairer comparison between the methods, to better identify their strengths and weaknesses.

Question 2. *What is the most significant drawback of current methods? Why do they not scale? What are they missing, and how can we address this?*

Answering Question 2 contributes to address Challenges 2, 4, and 4. Chapter 6 identifies the most impactful shortcomings of current CA methods, and Chapter 9 advances a proposal for a new method to address those.

Question 3. *Experiments are slow and bounded by the computational resources available. Can we improve the* ***evaluation*** *of credit assignment methods by improving the speed at which we can iterate on an algorithm? How? Is there a way to expedite the benchmarking of CA methods?*

Answering Question 3 contributes to address Challenge 5. Chapter 7 surveys the current protocols and environments to evaluate CA methods, and Chapter 10 proposes a new, high-throughput environment to expedite the benchmarking.

1.6 Goals, aims and scope

This dissertation proposes potential answers to the questions above, and sets out to realign the fundamental issue raised by Minsky (1961) to the Deep RL framework we know today. In addition, it prepares the future works around the CAP to intersect the latest developments in the broader AI field, such as high-throughput training, alignment, the development of a foundation model for RL, and the problem of AI safety.

Goals. The first goal is to provide an overview of the field to new-entry practitioners and researchers, and, for scholars looking to develop the field further, to put the Ì set of works into a comprehensive, coherent perspective. Lastly, we aim to reconnect works whose findings are relevant for CAP, but that do not refer to it directly. To the best of our knowledge, the work by Ferret (2022, Chapter 4) is the only effort in this direction, and the literature offers no explicit surveys on the temporal CA problem in Deep RL. This contributes to answer Question 1.

Having established the problem we aim to solve, we them aim to investigate why current methods do not scale, and how we can address this. We select two promising methods, *options* and *reward shaping*, and design a new method that transfers the prior knowledge of an LLM about a game into the agent's value function using reward shaping and options discovery. This contributes to answer Question 2.

Finally, we aim to improve the evaluation of credit assignment methods by improving the speed and the throughput at which we can iterate on an algorithm. This not only speeds up existing benchmarks, but due to the magnitude of the improvement in speed, it proposes a whole new paradigm for RL training, allowing to perform experiments that would be otherwise impossible due to the long time required to run them. This contributes to answer Question 3.

Scope. The thesis focuses on temporal CA in single-agent Deep RL, and the problems of *(i)* quantifying the influence of an action mathematically and formalising a mathematical objective for the CA problem *(ii)* defining its challenges, and categorising the existing methods to learn the quantities above, *(iii)* defining a suitable evaluation protocol to monitor the advancement of the field. We do not discuss *structural* CA in Deep Neural Networks (DNNs), that is, the problem of assigning credit or blame to individual parameters of a DNN (Schmidhuber, 2015; Balduzzi et al., 2015). We also do not discuss CA in multi-agent RL, that is, to ascertain which agents are responsible for creating good reinforcement signals (Chang et al., 2003; Foerster et al., 2018). When credit (assignment) is used without any preceding adjective, we always refer to *temporal* credit (assignment). In particular, with the adjective *temporal* we refer to the fact that *"each ultimate success is associated with a vast number of internal decisions"* (Minsky, 1961) and that these decisions, together with states and rewards, are arranged to form a *temporal* sequence.

The thesis focuses on Deep RL. In surveying existing formalisms and methods, we only look at the Deep RL literature, and when proposing new ones, we tailor them to Deep RL theories and applications. We exclude from the review methods specifically designed to solve decision problems with linear or tabular RL, as they do not bode well for scaling to complex problems. In designing new methods, we focus on a Deep RL framework, for its promise to scale with data and model size. In designing benchmarks and evaluation methods, we tailor them to Deep RL applications.

1.7 Outline, contributions, and publications

The dissertation is structured in three parts: **Contexts**, **Understanding**, and **Solving**. Each part is composed of chapters that address the research questions we set out to answer.

Part II (Contexts) provides the foundational concepts and the notation at the base of the manuscript. In particular, **Chapter 2** describes the required background on RL, and **Chapter 3** defines the set of assumptions for our studies.

Part III (Understanding) seeks for a better understanding of the problem. We provide an extensive survey of the field, which stands out as a novel contribution in itself and provides a novel, coherent formalism on the CAP. In particular, **Chapter 4** defines the assumption at the base of the problem, clarifies what is the credit of an action, how does the literature quantify it, and discusses interesting properties that a measure of credit should have. **Chapter 6** formalises a CA *method*, provides a taxonomy of existing methods and uses the formalism from **Chapter 4** to draw a fair comparison between them. In **Chapter 7** we review the protocols and benchmarks to evaluate a credit assignment method. This is necessary to be able to track progress, distinguish it from non-progress, and to define a shared protocol to monitor advancements in the field.

Having clearly defined the problem, **Part IV (Solving)** sets out to address the limitations of the solutions and evaluation methods.

In **Chapter 9**, we set out to address the limitations of some of the most promising methods to assign credit and scale RL to the real world: options and reward shaping. We go beyond methods that learn tabula rasa and design a new one that transfers the prior knowledge of an LLM about a game into the agent's value function using reward shaping and options discovery. The method provides a way to save the costs of human labelling, usually necessary for options and reward shaping, allowing to scale them to highly complex environments.

In **Chapter 10**, we look at the evaluation problem, and at how we can shorten the long iteration cycles of RL algorithm design. We propose NAVIX, a reimplementation of MiniGrid in JAX, that shortens experiment times from 1 week to 15 minutes. This allows not only faster algorithm verifications, enabling faster design iterations, but also it allows performing experiments that would be otherwise impossible due to the long time required to run them.

Contributions. This corpus provides the following contributions:

i. A comprehensive review of the field, providing a novel, coherent formalism for the CAP. We review and discuss how existing works quantify action influences in RL, propose a unifying formalism to compare them and identify the most appropriate contexts to use them.

ii. Based on this novel framework, a taxonomy of the existing methods to assign credit that brings together works that are relevant to the CAP, but that do not refer to it explicitly. This allows to draw a fair comparison between them.

iii. A review and discussion of the existing protocols and benchmarks to evaluate a CA method, highlighting their perks and drawbacks. The review is instrumental to set the stage for the development of a shared protocol to monitor advancements in the field.

iv. Overall, items *i.*, *ii.*, and *iii.* constitute a survey that stands out as a novel contribution in itself. To the best our knowledge, it is the first coherent survey on the topic since Minsky's seminal work in 1961, and proposes a view on the current gaps to close to advance the field and scale RL to very complex problems.

v. We propose a new method to assign credit that aims to bridge the gap between human and RL agents by transferring the prior knowledge of an LLM into the agent's value function via reward shaping and options discovery.

vi. Finally, we focus on the evaluation problem and propose a reimplementation of MiniGrid in JAX, that shortens experiment times from 1 week to 15 minutes.

Publications. The work in this dissertation has been published in the following venues:

- Eduardo Pignatelli, Johan Ferret, Matthieu Geist, Thomas Mesnard, Hado van Hasselt, and Laura Toni. A survey of temporal credit assignment in deep reinforcement learning. *Transactions on Machine Learning Research*, 2024a. ISSN 2835-8856. URL `https://openreview.net/forum?id=bNtr6SLgZf`. Survey Certification

- Eduardo Pignatelli, Johan Ferret, Davide Paglieri, Samuel Coward, Tim Rocktäschel, Edward Grefenstette, and Laura Toni. Assessing the zero-shot capabilities of LLMs for action evaluation in RL. In *ICML Workshop on Automated Reinforcement Learning*, 2024b. URL `https://openreview.net/forum?id=MFw8K5705I`

- Eduardo Pignatelli, Jarek Liesen, Robert Tjarko Lange, Chris Lu, Pablo Samuel Castro, and Laura Toni. Navix: Scaling minigrid environments with jax. *arXiv preprint arXiv:2407.19396*, 2024c. In submission to ICML 2025

Part II

Background

Chapter 2

Reinforcement Learning

We owe to Sutton and Barto (Sutton & Barto, 2018), who have drawn on the research by Klopf (Klopf, 1972; 1975; 1982; 1988) and Watkins (Watkins, 1989; Watkins & Dayan, 1992), the systematisation of RL into the computational framework we acknowledge today. This chapter describes this formal approach – the reader familiar with this paradigm can skip this chapter, except for Section 2.1.

2.1 Notations

We use calligraphic characters to denote sets and the corresponding lowercases to denote their elements, for example, $x \in \mathcal{X}$. For a measurable space $(\mathcal{X}, \Sigma)$, we denote the set of probability measures over $\mathcal{X}$ with $\Delta(\mathcal{X})$. We use an uppercase letter X to indicate a random variable, and the notation $\mathbb{P}_X$ to denote its distribution over the sample set $\mathcal{X}$, for example, $\mathbb{P}_X : \mathcal{X} \to \Delta(\mathcal{X})$. When we mention a *random event* X (for example, a *random action*) we refer to a random draw of a specific value $x \in \mathcal{X}$ from its distribution $\mathbb{P}_X$ and we write, $X \sim \mathbb{P}_X$. When a distribution is clear from the context, we omit it from the subscript and write $\mathbb{P}(X)$ instead of $\mathbb{P}_X(X)$. We use $\mathbb{1}_{\mathcal{Y}}(x)$ for the indicator function that maps an element $x \in \mathcal{X}$ to 1 if $x \in \mathcal{Y} \subset \mathcal{X}$ and 0 otherwise. We use $\mathbb{R}$ to denote the set of real numbers and $\mathbb{B} = \{0, 1\}$ to denote the Boolean domain. We use $\ell_\infty(x) = \|x\|_\infty = \sup_i |x_i|$ to denote the ℓ-infinity norm of a vector x of components x_i. We write the Kullback-Leibler divergence between two discrete probability distributions $\mathbb{P}_P(X)$ and $\mathbb{P}_Q(X)$ with sample space $\mathcal{X}$ as: $D_{KL}(\mathbb{P}_P(X)||\mathbb{P}_Q(X)) = \sum_{x\in\mathcal{X}}[\mathbb{P}_P(x)\log(\mathbb{P}_P(x)/\mathbb{P}_Q(x))]$.

2.2 Markov decision processes

MDPs formalise decision-making problems. This thesis focuses on the most common MDP settings for Deep RL. Formally, a discounted MDP (Howard, 1960; Puterman, 2014) is defined by a tuple

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, \mu, \gamma)\,. \tag{2.1}$$

$\mathcal{S}$ is a finite set of states (the *state space*) and $\mathcal{A}$ is a finite set of actions (*the action space*). $R : \mathcal{S} \times \mathcal{A} \to [r_{min}, r_{max}]$ is a deterministic, bounded reward function that maps a state-action pair to a scalar reward r. $\gamma \in [0, 1]$ is a discount factor and $\mu : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is a transition kernel, which maps a state-action pair to probabilities over states. We refer to an arbitrary state $s \in \mathcal{S}$ with s, an action $a \in \mathcal{A}$ with a and a reward $r \in [r_{min}, r_{max}]$ with r. Given a state-action tuple (s, a), the probability of the next random state S_{t+1} being s' depends on a *state-transition* distribution: $\mathbb{P}_\mu(S_{t+1} = s'|S_t = s, A_t = a) = \mu(s'|s, a), \forall s, s' \in \mathcal{S}$. We refer to S_t as the *random state* at time t. The probability of the action a depends on the agent's policy, which is a stationary mapping $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, from a state to a probability distribution over actions.

These settings give rise to a discrete-time, stateless (Markovian), Random Process (RP) with the additional notions of *actions* to represent decisions and *rewards* for a feedback signal. Given an initial state distribution $\mathbb{P}_{\mu_0}(S_0)$, the process begins with a random state $s_0 \sim \mathbb{P}_{\mu_0}$. Starting from s_0, at each time t the agent interacts with the environment by choosing an action $A_t \sim \mathbb{P}_\pi(\cdot|s_t)$, observing the reward $r_t \sim R_t(S_t, A_t)$ and the next state $s_{t+1} \sim \mathbb{P}_\mu$. If a state s_t

is also an *absorbing* state ($s \in \overline{\mathcal{S}} \subset \mathcal{S}$), the MDP transitions to the same state s_t with probability 1 and reward 0, and we say that the episode terminates. We refer to the union of each temporal *transition* (s_t, a_t, r_t, s_{t+1}) as a *trajectory* or *episode* $d = \{s_t, a_t, r_t, : 0 \leq t \leq T\}$, where T is the *horizon* of the episode.

We mostly consider episodic settings where the probability of ending in an absorbing state in finite time is 1, resulting in the random horizon T. We consider discrete action spaces $\mathcal{A} = \{a_i : 1 \leq i \leq n\}$ only.

A trajectory is also a random variable in the space of all trajectories $\mathcal{D} = (\mathcal{S} \times \mathcal{A} \times \mathcal{R})^T$, and its distribution is the joint of all of its components $\mathbb{P}_D(D) = \mathbb{P}_{A,S,R}(s_0, a_1, r_1, \ldots, s_T)$. Given an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, \mu, \gamma)$ and fixing a policy π produces a Markov Process (MP) $\mathcal{M}^\pi$ and induces a distribution over trajectory $\mathbb{P}_{\mu,\pi}(D)$.

We refer to the *return* random variable Z_t as the sum of discounted rewards from time t to the end of the episode

$$Z_t = \sum_{k=t}^{T} \gamma^{k-t} R(S_k, A_k). \tag{2.2}$$

The *control* objective of an RL problem is to find a policy π^* that maximises the expected return,

$$\pi^* \in \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{\mu,\pi} \left[\sum_{t=0}^{T} \gamma^t R(S_t, A_t) \right] = \mathbb{E}\left[Z_0\right]. \tag{2.3}$$

MDPs are abstractions. Because MDPs are considerable abstractions of decision problems, decisions are modelled as abstract concepts, and their meaning can arbitrarily vary in scale, from low to high level acts. For example, the coordinated choreography of decisions performed to "grab a glass" can be broken down either into a sequence of small muscular movements, such as "bend the bicep, stretch the triceps", or higher level intentions, such as "extend the arm, rotate the wrist", as well as a standalone action itself. This flexibility allows treating the anecdotal situations described in the introduction with investigative rigour and precise theoretical statements. In fact, many fundamental theorems underpinning RL solutions assume the formulation of an MDP.

What does Markovian mean? Stateless processes respect the *Markov* property: the future is independent of the past, and it is determined only by the present state of the system. In other words, the present holds all the information from the history that is required to propagate the current state further in time. Once the present is known, the history can be disposed of. Given the present, the probability that a particular *state-reward* event occurs in the future depends on the specification of a *state-transition* distribution. Sampling from this distribution yields a temporal sequence composed of intertwined packets – states, actions, and rewards $\langle s_t, a_t, r_t, \rangle$ – the *trajectory*.

Episodic and continuing settings. An MDP, and so a trajectory, can be finite or infinite in time. There exists a particular subset of states, whose only possible transition is to themselves with probability 1: the set of *absorbing* states. Absorbing states define the *termination* of a decision process, and their reward is, by definition, always zero. Termination is a natural notion for tasks that intuitively break down into segments of time, such as the level or the turn of a game. In these settings, the trajectory takes the name of *episode*, and the task, the adjective *episodic*. Each episode is considered an independent compartment of experience and information across episodes may, or may not be shared. There exists the analogous notion for decision problems that are intuitively infinite, named *continuing* tasks. Their formal treatment presents additional problems, except, because it is possible to unify their notation, we defer to (Sutton & Barto, 2018) for details, and assume in the rest of the manuscript that any task is an episodic task.

2.2.1 Policies

In Section 1.1, we introduced the concept of *policy*, as a *rule of behaviour*, here we formalise it. By formatting experience into different rules of behaviour, the agent can tune its performance and modulate its interactions with the environment. Formally, a policy $\pi(s)$ that spans a policy space Π, is a mapping from a single state to probabilities over actions.

Since a policy reacts to each observation with an action programmatically, it modifies the probability to visit a certain state. In other words, policies induce a state-visitation probability, denoted as $\rho_\pi(s)$, on top of the state-transition probability distribution, altering the likelihood of sampling a specific state.

$$\rho_\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \mathbb{P}_\mu(s_t = s|\pi) \tag{2.4}$$

Because the expected sequence of rewards depends on the state-visitation probability, the return is an intrinsic property of the policy as well as of the MDP.

There are many ways to represent policies, depending on the problem setup. Often, when the state space or the action space are large, policies are parameterised with a vector θ with dimensionality smaller than the number of all possible states. Since we assume $\dim(\theta) < |\mathcal{S}|$, the parameterised policy cannot represent all behaviours. The set of policies Π is restricted to include only the set of representable policies $\Pi^\theta \subset \Pi$. Different parameterisation techniques, e.g., tables, linear polynomials, or deep neural networks, yield different declensions of RL, respectively *tabular*, *linear*, and *deep* RL. Sutton (Sutton & Barto, 2018) offers extensive material on tabular and linear methods, while we briefly describe deep RL in Section 2.7.

2.3 Partially-observable Markov decision processes

Partially Observable Markov Decision Processes (POMDPs) are a generalisation of MDPs introduced to relax the assumption of complete information to one of *incomplete information* (Åström, 1965; Kaelbling et al., 1998). In a POMDP, the system dynamics are governed by an underlying Hidden Markov Model, except that the agent cannot access the full MDP state. The partially observed state is the result of a mapping from true state to *observation*: the *observation function*. Because the agent cannot access the true present, it must form its own belief on the environment, and take its decisions under uncertainty.

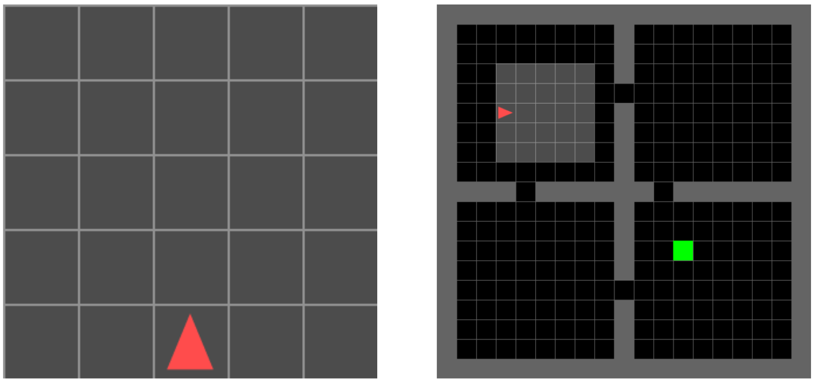
Formally, a POMDP is defined by the tuple:

$$\mathcal{E} = \langle \mathcal{S}, P, \mathcal{A}, \mathcal{R}, \gamma, O, \mathcal{O} \rangle \tag{2.5}$$

Here, the tuple $\langle \mathcal{S}, P, \mathcal{A}, \mathcal{R}, \gamma \rangle$ is the same as for the definition of the MDP. O is the set of all observations and $\mathcal{O}$ is the observation function, where $\mathbb{P}_{\mathcal{O}}(o_{t+1}, r_{t+1} | o_t, a_t)$ is the probability of reading the observation-reward pair $\langle o_{t+1}, r_{t+1} \rangle$, given the observation-action pair $\langle o_t, a_t \rangle$. It is not uncommon to use the terms *state* and *observation* interchangeably, while inferring the exact meaning from context: the type of decision process involved. This is also the notation and terminology we use in this manuscript.

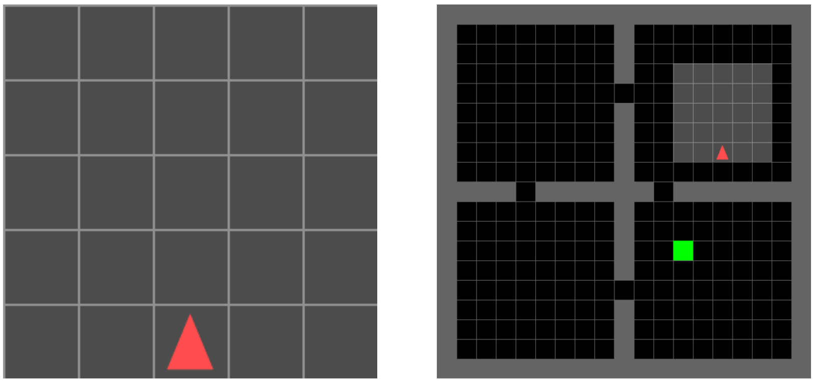
2.3.1 State aliasing

While in MDPs a state can unequivocally define the present, this is not the case for POMDPs. *State aliasing* is a common symptom of partially observable environments, and occurs when two or more distinct states degenerate into the same observation. Two states $s_1, s_2 \in \mathcal{S}$ are said to be aliased in the



(a) (b)

Figure 2.1: The four-room GridWorld (Chevalier-Boisvert et al., 2018) is a popular testbed for RL algorithms. The task is for the agent – the red arrow – to navigate to a goal position – the green square. The agent can choose among the following actions: i) turn counterclockwise; ii) turn clockwise; iii) move forward; iv) pick up an object; v) drop an object; vi) toggle object; vii) wait. **(a)** The observation of this POMDP is a partial RGB view of the environment centred around the agent's perspective. **(b)** The underlying MDP state, showing the agent's global position. The lighter area corresponds to the agent's field of view, while the rest is considered darkness or not visible.



(a) (b)

Figure 2.2: An example of state aliasing. The figure shows the same POMDP described in Figure 2.1, but in a different MDP state, and its corresponding observation. Notice how the observation in **(b)** is exactly the same as the observation in Figure 2.1(a), while the underlying MDP state is different. While the optimal action in Figure 2.1(b) would be to move forward, it is to turn counterclockwise for the state in **(b)**. However, this information is not available to the agent, which will receive the same observations Figure 2.1(b) and **(b)**.

observation $o = o_1 = o_2 \in O$ if:

$$\begin{cases} o_1 = \mathcal{O}(s_1) \\ o_2 = \mathcal{O}(s_2) \\ s_1 \neq s_2 \\ o_1 = o_2 = o \end{cases} \tag{2.6}$$

Because the policy in an MDP is a mapping from observation to action, when two states are aliased into one observation, but they attend to two different optimal actions, the policy does not have enough information to discern between the two. It therefore assigns, suboptimally, the same action to two different underlying states. For example, consider the task – shown in Figure 2.1 and Figure 2.2– of navigating a partially dark room, with the objective to reach the desired goal. When standing in the middle of any room, darkness does not allow to tell the global direction towards which we are facing, since all around us looks exactly the same. However, if we are facing east – such as in Figure 2.1 right – the optimal action would be to turn around by 90 degrees clockwise, and then swing ahead. Instead, in case we face south – such as in Figure 2.2 right – the same policy results in suboptimal behaviour, for it points to the wall. Notice that what we see is exactly the same in both cases, and only additional information can help us differentiate between the two, and turn our belief state into the true state of the underlying decision problem. One way to complement this information is to consider the history of observations, whatever long. For example, remembering where we come from could be enough. For this reason, while MDP policies are a mapping from a single state to action probabilities, a POMDP policy is a mapping from a *history* of observations to probabilities of actions. The history of observations constitutes a single state of our decision problem, and policies update accordingly:

$$\pi(x) = \mathbb{P}_\pi(a|x) \tag{2.7}$$

where $x = \{o_t, o_{t+1}..., o_{t+k}\}$ is the history of observations; $k \in \mathbb{N}$ is an arbitrary length of the history, and $k = 0$ recovers the definition of policy in an MDP, given that $\mathcal{O}(s) = s$.

2.4 The value problem

As detailed in the previous section, different policies acting in an environment define different trajectory random variables, and, in turn, different returns. The concept of utility, borrowed from economics and optimal control, proposes a *model* of the returns – the *value* – that allows modelling *preferences* mathematically. In RL, the *value* of a state is the device that formalises the notion of *utility*, and a central quantity. Given a policy π, the *state value function* maps a starting state s_t to its expected return G_t. Intuitively, values measure how much reward the agent can expect to collect in the future, thus allowing to evaluate policies. Formally, the value of a state is defined as:

$$v_\pi(s) = \mathop{\mathbb{E}}_{\rho}\left[G_t | S_t = s\right], \quad \forall s \in \mathcal{S} \tag{2.8}$$

Here, the expectation $\mathbb{E}_\rho$ is over the state visitation distribution, comprising the environment dynamics μ, and the stochasticity of policy π.

Since value functions $v(s)$ are models of returns $G(s)$, they can be estimated from experience following a process called *policy evaluation*. One fundamental property of returns and values is that they satisfy a recursive relationship in time. By the law of iterated expectations, one can write the value at time $v(s_t)$ by referring to the value of its immediate successor state $v(s_{t+1})$. In particular, the value of a state $v(s_t)$ is, in expectation, equal to the reward collected after taking an action $a \sim \pi(s_t)$, plus the discounted value of the next state $v(s_{t+1})$. This relationship yields one of the most central equations in RL: the Bellman

Expectation Equation.

$$
\begin{aligned}
v_\pi(s_t) &= \mathop{\mathbb{E}}_{\pi,\mu}\left[r_t + \gamma v_\pi(s_{t+1})\right] && (2.9)\\
&= \sum_a \pi(a|s) \sum_{s_{t+1},r} \mathbb{P}_\mu(s_{t+1}, r|s_t, a)\left[r_t + \gamma v_\pi(s_{t+1})\right] && (2.10)\\
&= \rho_\pi\left[r_t + \gamma v_\pi(s_{t+1})\right]. && (2.11)
\end{aligned}
$$

Nonetheless, despite state values being a useful metric to evaluate a policy, they do not provide information on how to value individual actions. There exists a second type of value function, called the *state-action value* function, which represents the value of taking a particular action a_t when starting from a state s_t, and following π afterwards. State-action values are a key quantity to define and solve the optimal policy problem:

$$
q_\pi(s,a) = \mathop{\mathbb{E}}_{\pi,\mu}\left[G_t|S_t = s, A_t = a\right], \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \tag{2.12}
$$

Importantly, they satisfy an analogous recursive relationship, defining the Bellman Expectation Equation for state-action values:

$$
\begin{aligned}
q_\pi(s_t, a_t) &= \mathop{\mathbb{E}}_{\pi,\mu}\left[r_t + \gamma q_\pi(s_{t+1}|\pi(s_{t+1})\right]\\
&= \sum_a \pi(a|s) \sum_{s_{t+1},r} \mathbb{P}_\mu(s_{t+1}, r|s_t, a)\left[r_t + \gamma q_\pi(s_{t+1}|\pi(s_{t+1})\right] && (2.13)\\
&= \rho_\pi\left[r_t + \gamma q_\pi(s_{t+1}|\pi(s_{t+1})\right]
\end{aligned}
$$

Finally, a third quantity, function of the previous two values, is of high interest for credit assignment: the *advantage* function. Advantages measure how convenient it is to take one action a with respect to the other actions in the action set $\mathcal{A}$. Its importance is paramount since, by providing a relative measure among actions, it requires knowledge about the values of *all* actions in the action set. Arguably, this is the quantity we care the most about, but in practice, estimating it is far more expensive. Formally:

$$k_\pi(s,a) = q_\pi(s,a) - v_\pi(s) \tag{2.14}$$

Here, $k_\pi(s,a)$ is the advantage function of the policy π, and $q_\pi(s,a)$ and $v_\pi(s)$ are the state-action and state value functions, respectively.

2.5 The optimal policy problem

Solving a RL problem means finding a policy that collects the maximal return G, in expectation[1]. Value functions are at the core of this problem, for their capacity to represent expected returns. RL exploits three fundamental properties of value functions.

(I) **Ordering.** Value functions induce a partial ordering over policies. A policy π is said to be greater (better) than or equal to another policy $\pi^{'}$ if and only if its expected return is greater than or equal to that of $\pi^{'}$:

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s), \quad \forall s \in \mathcal{S}. \tag{2.15}$$

(II) **Optimal set.** Secondly, in the partially ordered set Π^θ there is at least one policy that is greater than or equal to all other policies in the representable set. This set of policies is called the *set of optimal policies*, denoted as Π^*, whereas any optimal policy is represented as π^*. Even if there is no guarantee that Π^* is a singleton, all optimal policies evaluate to a unique, shared value function, called the *optimal value function.* Formally one can write:

$$v^*(s) = v^{\pi^*}(s) = \max_\pi v_\pi(s) \qquad \forall s \in \mathcal{S} \tag{2.16}$$

$$q^*(s,a) = q^{\pi^*}(s,a) = \max_\pi q_\pi(s,a) \qquad \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \tag{2.17}$$

[1]There are also works aiming to maximise the return random variable (Bellemare et al., 2017; Dabney et al., 2018).

where $v^*(s)$ and $q^*(s,a)$ are the optimal state and state-action value functions, respectively. Intuitively, the optimal state value function is the state-action value where the action has the highest value:

$$v^{\pi^*}(s) = \max_a \left[q^{\pi^*}(s,a)\right], \quad \forall a \in \mathcal{A} \tag{2.18}$$

Because optimal value functions respect the same recursive relationship of any value function, substituting (2.9) into (2.18) leads to:

$$v^*(s) = \max_a \mathop{\mathbb{E}}_{\pi,\mu} \left[r_t + \gamma v^*(s_{t+1})\right] \tag{2.19}$$

$$q^*(s,a) = \max_a \mathop{\mathbb{E}}_{\pi,\mu} \left[r_t + \gamma q^*(s_{t+1}, \pi^*(s_{t+1})\right] \tag{2.20}$$

The arrangements of equations (2.16) and (2.17) into equations (2.19) and (2.20) constitute the Bellman Optimality Equations for, state and state-action value functions, respectively.

(III) **Existence.** Finally, notice that not every pair of policies (value functions) is comparable because the set of policies (value functions), is only *partially* ordered. Two policies π and $\pi^{'}$ can be in only one of the following four relationships: i) $\pi > \pi^{'}$; ii) $\pi < \pi^{'}$; iii) $\pi = \pi^{'}$; finally iv) $\pi \not\gtrless \pi^{'}$, where $\not\gtrless$ denotes non-comparable items. This is overt if the value of a policy $v_\pi(s)$ is greater than the value of a second policy $v_{\pi'}(s)$ in some states, but not in others. If not all policies can be compared, how can we be sure that an optimal policy even exists? In other words, that $\Pi^* \neq \emptyset$? None of the above formulations have yet provided a mathematical proof of the intuition. RL theory resorts to the Banach fixed-point theorem to prove that the Bellman Optimality operator $\mathcal{T} : \Pi \to \Pi$ is a contraction mapping in the metric space (Π, v), and its unique fixed point ensures the existence of one optimal value function. Analogously, the existence of at least one optimal policy is always guaranteed by the existence of an optimal value function. Also, notice that the uniqueness of optimal policies is not guaranteed and the set of optimal policies might not be a singleton. This distinction is key for the concept of credit, since not all optimal policies assign credit optimally. The next section discusses solutions for optimal policies and optimal value function.

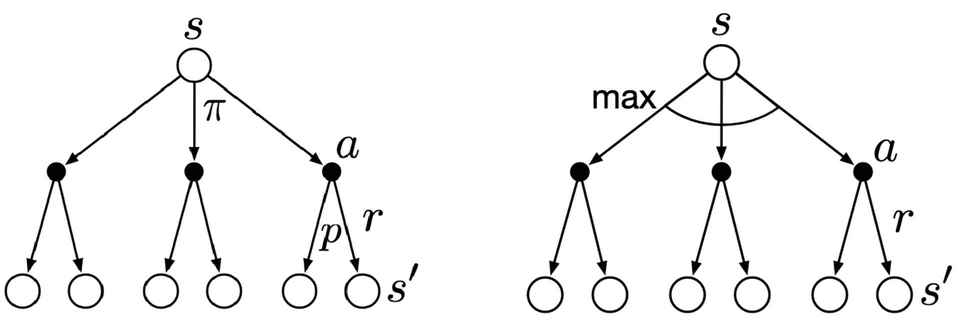
2.6 Solving the optimal policy problem

Before the discussion delves into a taxonomy of methods for solving prediction and control problems, we zoom-out to reconnect with the broader problem of decision making. For an artificial agent, we identify tree main sources of knowledge [2].

Priors. Priors constitute the non-acquired knowledge deriving from a particular configuration of starting conditions. This kind of knowledge can be interpreted as the reduction of uncertainty about the environment that is not conditioned to any experience. A key, denoting factor of priors is that they are a property of one specific *instance* of the agency (e.g., yourself), and not of a class of agencies (e.g., humans). Genotypes are one example of priors in biology, for they define the starting point of human knowledge accumulation. The equivalent notion in machine learning is initialisations.

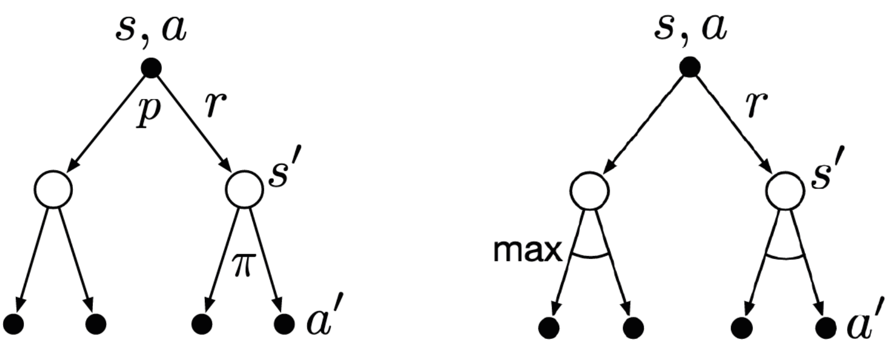
Experience. Experience is at the core of any learning systems. It can be interpreted as the perpetual digestion of external sensory data into general, more abstract statements about the world. As a result, all else being the same, different experiences strongly influence what knowledge an agent possesses. Curricula are a clear example of this interaction. For example, canonical education presents additions before multiplications, and equations, and, again, linear algebra. Experiencing these concepts backwards would, most likely, result in weaker learning experiences. In RL, this phenomenon is not only particularly relevant, but also a trait that distinguishes this form of learning

[2] There exist many definitions of knowledge and its definition is not a subject of this dissertation. Here we assume a, probably simpler, information theoretic definition of knowledge using entropy: the reduction of uncertainty about a random variable. The random variable we refer to is the environment.



(a) (b)

Figure 2.3: Backup diagrams for a state value function update. Hollow circles signify states, whereas solid circles denote actions. (a) State value – $v(s)$ – backup to evaluate a policy according to the Bellman expectation equation. (b) State value backup for the optimal state value function – $v^*(s)$. The arc over the branches represents a maximisation operation. Here s is the state, π is the policy, a is a potential action, r is the reward, p is the state-transition probability and $s^{'}$ is a possible next state.



(a) (b)

Figure 2.4: Backup diagrams for a state-action value function update. Hollow circles signify states, whereas solid circles denote actions. (a) State-action value backup – $q(s)$ – to evaluate a policy according to the Bellman expectation equation. (b) State-action value backup for the optimal state-action value function – $q^*(s)$. The arc over the branches represents a maximisation operation. Here s is the state, π is the policy, a is a potential action, r is the reward, p is the state-transition probability and $s^{'}$ is a possible next state. Notice how the root of the backup tree in a state-action value function is a state-action pair, unlike in a state value backup. Also notice how, after evaluating the chosen action, the following action is sampled from $\pi(s^{'})$. This allows to evaluate "how good" is the action a, when the agent is in a state s.

from others, such as supervised learning: the decisions of the present strongly affect the parts of the world experienced in the future.

Learning biases. Learning or *inductive* biases, are the assumptions at the base of a *type* of learners. Unlike priors, biases are a property of a class of agencies (e.g., humans), usually fixed by some higher order mechanism, such as evolution, or scientific innovation. Inductive biases include learning mechanism, and describe how experience is baked into knowledge. They also include perceptual components, such as the acquisition, storage, processing, and retrieval of experience. In humans, the neuroscientific literature offers numerous models of learning biases, e.g., dopaminergic metabolism for goal-oriented behaviour. In RL, inductive biases strongly influence credit assignment: how is experience transformed into knowledge? How is it transferred from the external world into a different order of parameters?

The remainder of this section describes the fundamental methods to solve the optimal policy problem. We start by introducing the policy iteration schema, and then branch out to two main classes of algorithms: temporal difference learning and policy gradient methods. Finally, we describe how these adapt to approximate settings where knowledge acquisition is limited, and briefly illustrate the notion of Deep RL.

2.6.1 Generalised policy iteration

Solving RL problems most often means to assume a set of *learning biases* and to use *experience* to adapt *prior* knowledge to decide optimally. Combinations of different learning biases (Mnih et al., 2015; Schulman et al., 2015; 2017; Bellemare et al., 2017; Haarnoja et al., 2018), priors (Finn et al., 2017; Javed & White, 2019) and experiences (Bellemare et al., 2013; Cobbe et al., 2020; Samvelyan et al., 2021; Todorov et al., 2012; Jiang et al., 2021a) yield a sheer variety of algorithms. Mathematically, they all follow the Generalised Policy Iteration (GPI) schema. Policy Iteration is the process that initiates from an arbitrary policy and then exploits experience to alternate between steps of *policy evaluation* and *policy improvement*.

Policy Evaluation (PE) is then the process that maps a policy π to its value function. A canonical PE procedure starts from an arbitrary value function V_0 and iteratively applies the Bellman expectation operator, Λ, such that:

$$\hat{v}^{\pi}_{k+1}(S_t) = \Lambda^{\pi}[\hat{v}^{\pi}_{k}(S_t)] := \mathbb{E}_{\pi,\mu}\left[R(S_t, A_t) + \gamma \hat{v}_k(S_{t+1})\right], \tag{2.21}$$

where $\hat{v}_k$ denotes the value approximation at iteration k, $A_t \sim \mathbb{P}_{\pi}(\cdot|S_t)$, and $S_{t+1} \sim \mathbb{P}_{\mu,\pi}(\cdot|S_t, A_t)$. The Bellman operator is a γ-contraction in the ℓ_∞ and the ℓ_2 norms, and its fixed point is the value of the policy π. Hence, successive applications of the Bellman expectation operator improve the prediction accuracy because the current value gets closer to the true value of the policy. We refer to the PE as the *prediction* objective (Sutton & Barto, 2018).

Policy improvement is the process of applying the Bellman optimality operator Γ, maps a policy π to an improved policy:

$$\pi_{k+1}(a|S) = \Gamma[\pi_k, S] = \mathbb{1}_{\{a\}}(\operatorname*{argmax}_{u\in\mathcal{A}}\left[R(S,u) + \gamma v_k(S')\right]) = \mathbb{1}_{\{a\}}(\operatorname*{argmax}_{u\in\mathcal{A}}\left[q_k(S,u)\right]). \tag{2.22}$$

Notice how an improvement step generates a new policy, which, in turn, needs to be re-evaluated. Only when the current policy is optimal the two processes agree to convergence. The Bellman optimality equation will produce the same policy, and the expectation equation will not need to re-evaluate, as the value function can already tell the returns of the current policy. In theory, the process stops when both equations produce no changes: a fixed-point of both the Bellman expectation and optimality equations. In practice, we often terminate when policy improvement touches a stationary point.

We refer to GPI as a general method to solve the *control* problem (Sutton &

Barto, 2018) deriving from the composition of PE and Policy Improvement (PI). In particular, we refer to the algorithm that alternates an arbitrary number k of PE steps and one PI step as Modified Policy Iteration (MPI) (Puterman & Shin, 1978; Scherrer et al., 2015). For $k = 1$, MPI recovers Value Iteration, while for $k \to +\infty$, it recovers Policy Iteration. For any value of $k \in [1, +\infty)$, and under mild assumptions, MPI converges to an optimal policy (Puterman, 2014).

Different specifications of the evaluation and improvement processes produce different families of algorithms. The next two sections describe the main categories of algorithms, separated by whether they explicitly represent the policy or not: action-value methods and policy gradients.

2.6.2 Action-value methods

Action-value methods include the class of techniques that consult state-action value functions to select suitable actions. At the foundation of these methods is Temporal difference learning (Samuel, 1959; Klopf, 1972). Here the value estimate is updated at every new incoming experience, by measuring the difference between the current inference, and the actual, experienced outcome. The actual (partial or not) outcome is called *value target*, and different target specifications bring different action-value algorithms. A generic step of policy improvement in action-value methods can be described as:

$$v^{\pi}_{i+1}(s_t) = v^{\pi}_{i}(s_t) + \alpha \left[\hat{G}(s_t) - v^{\pi}_{i}(s_t)\right] \tag{2.23}$$

where $v^{\pi}_{i+1}(s_t)$ is the value function under the policy π, at iteration $i+1$ and state s_t; $v^{\pi}_{i}(s_t)$ is the value at iteration i; α is a step-size parameters to control the importance of the sampled experience s_t; $\hat{G}(s_t)$ is the *value target*. The quantity $\delta_t = \hat{G}(s_t) - v^{\pi}_{i}(s_t)$ takes the name of *temporal difference error*.

Depending on the value target, there exists a spectrum of algorithms, each with their convergence properties. We refer the set of properties proposed by Rowland (Rowland et al., 2020): fixed-point bias, contraction rate, and variance. If the value target is the actual return of the full episode, we obtain the Monte-Carlo methods for both prediction and control.

$$v^{\pi}_{i+1}(s_t) = v^{\pi}_{i}(s_t) + \alpha \left[G(s_t) - v^{\pi}_{i}(s_t) \right] \tag{2.24}$$

where $\hat{G}(s_t) = G(s_t)$. Monte-Carlo methods are unbiased. They converge to the optimal policy exactly because the value target is the true return, but produce learning routines with high variance.

On the other extreme, $TD(0)$ only considers the immediate next transition step to update its target, and accounts for the future expected sum of rewards by adding the current estimate of the value function for the next state:

$$v^{\pi}_{i+1}(s_t) = v^{\pi}_{i}(s_t) + \alpha \left[r_t + \gamma v^{\pi}_{i}(s_{t+1}) - v^{\pi}_{i}(s_t) \right] \tag{2.25}$$

where $\hat{G}(s_t) = r_t + \gamma v^{\pi}_{i}(s_{t+1})$. Unlike Monte-Carlo methods, $TD(0)$ has lower variance, but its update targets might be biased with respect to the true return of the episode. In other words, the agent trades a slower process of knowing the effect of the current action on the most remote outcome of the episode in the future with a quick, one-step update based on limited knowledge. This introduces the fundamental concept of *bootstrapping*. Learning from incomplete episodes causes to formulate a guess from a guess: to produce expectations over future values from expectations of current ones.

Between these two extremes there exists a gamma of targets that blends the gap between Monte-Carlo and $TD(0)$ methods. In particular, two main techniques – *n-step bootstrapping* and *eligibility traces* – improve the value target by acting on two different dimensions of experience, respectively, time and memory. n-step bootstrapping works by considering a fixed n-step window to calculate the target. The agent starts in a state s_t, takes n steps following policy $\pi(s_t)$, calculates the return, and finally updates its policy:

$$v_{i+1}^{\pi}(s_t) = v_i^{\pi}(s_t) + \alpha \left[G_{t:t+n} - v_i^{\pi}(s_t)\right] \tag{2.26}$$

where:

$$G_{t:t+n} = \sum_{k=0}^{n-1} \left[\gamma^k r_{t+k}\right] + \gamma^n v_i^{\pi}(s_{t+n}) \tag{2.27}$$

By looking further into the future, the return of the starting state s_t is more accurate than a method exploiting a smaller time window and the effects of bootstrapping are less pronounced, but it is not as accurate as the full Monte-Carlo return. Furthermore, all n actions taken along the trajectory are updated, increasing the amount of information learned from one sample at the cost of reducing the frequency of each update Overall, the main advantage of n-step methods is the ability to trade off the bias caused by the bootstrapping with variance brought by long trajectories.

Eligibility traces, on the other side, calibrate the importance of an update based on a frequentist measure of the probability of a state-action event to happen. The intuition behind them is that the more an event is likely to happen, the more it is important to update its value estimate. The *trace* marks certain parameters of the policy, and registers their importance for the update. λ-*returns* are the core quantity under this view of eligibility traces, although there exist also different formulations:

$$v_{i+1}^{\pi}(s_t) = v_i^{\pi}(s_t) + \alpha \left[G_t^{\lambda} - v_i^{\pi}(s_t)\right] \tag{2.28}$$

where:

$$G_t^{\lambda} = (1-\lambda) \sum_{k=1}^{T} \lambda^{k-1} G_t \tag{2.29}$$

Notice how also *λ-returns* blend the gap between Monte-Carlo and one-step methods, but differ from n-step bootstrapping in the way they reduce the variance. While n-step returns decrease the accumulation window of the target, λ-returns re-weigh past returns exponentially less than newer ones.

Finally, n-step bootstrapping and λ-returns can be composed together to obtain an algorithm with the virtues of both techniques: the *truncated λ-return*:

$$v_{i+1}^{\pi}(s_t) = v_i^{\pi}(s_t) + \alpha \left[G_{t:t+n}^{\lambda} - v_i^{\pi}(s_t)\right] \tag{2.30}$$

where:

$$G_{t:t+n}^{\lambda} = \lambda^{n-1} G_{t:t+n} + (1-\lambda) \sum_{k=0}^{n-1} \lambda^{k-1} G_{t:t+k} \tag{2.31}$$

To summarise, action-value methods specify the policy improvement step by defining a target and acting greedily with respect to it. In the process, these methods must consult a value function to choose an action. In the next section we analyse a different approach to policy improvement, based on explicitly parameterising a policy and avoid consulting a value function.

2.6.3 Policy Gradient methods

Action-value methods work by searching for the optimal value function and then acting greedily with respect to it. Another approach to policy improvement is to explicitly parameterise the policy, and then directly search in the policy space. A broadly used family of method are *policy gradient methods*. Consider a differentiable policy $\pi(a|s,\theta)$, parameterised with a parameter vector θ. Policy gradient methods maximise the agent's performance by performing *gradient ascent* on $\pi(a|s,\theta)$, in the direction of the return target $\hat{G}$:

$$\theta_{i+1} = \theta_i + \alpha \frac{\partial \hat{G}(s)}{\partial \theta} \tag{2.32}$$

Here θ_{i+1} is the policy parameter vector at iteration $i+1$, and θ_i follows; α is a step-size parameter to ensure the convergence of the gradient descent; $\frac{\partial \hat{G}(s)}{\partial \theta}$ is the gradient of the maximisation objective $\hat{G}$ with respect to the policy parameters θ. One key obstacle of policy gradient methods is that the gradient $\hat{G}(s)$ depends on the environment and requires knowledge of the state-transitions distribution $\mu(s', r|s, a)$, or an approximate model of it. The *policy gradient theorem* overcomes this problem by removing the dependency on μ, and provides an analytical expression of the policy gradient without an explicit reference to the environment:

$$\frac{\partial \hat{G}(s)}{\partial \theta} = \mathbb{E}_\pi \left[\hat{G}(s) \gamma \frac{\partial \pi(a|s, \theta)}{\partial \theta} \right] \tag{2.33}$$

As usual, different specification of $\hat{G}(s)$ produce different algorithms. For example, considering the full Monte-Carlo return with $\hat{G}(s) = G(s)$, yields the *Monte-Carlo Policy Gradient*, or *REINFORCE* Williams (1992), update:

$$\theta_{i+1} = \theta_i + \frac{\alpha}{\pi(a_t|s_t, \theta)} \left[G(s_t) \gamma \frac{\partial \pi(a_t|s_t, \theta)}{\partial \theta} \right] \tag{2.34}$$

Substituting the score function $\frac{\nabla x}{x} = \nabla \log x$ one gets the most common form of policy gradient updates:

$$\theta_{i+1} = \theta_i + \alpha G(s_t) \gamma \nabla^\theta \log \pi(a_t|s_t, \theta) \tag{2.35}$$

where $\nabla^\theta \log \pi(a|s, \theta)$ is another expression for $\frac{\partial \log \pi(a_t|s_t, \theta)}{\partial \theta}$. A key advantage of policy gradient methods is the ability to represent stochastic policies, unlike most action-value methods. One natural way to define parametric policies is to use parameterised distributions. Softmax distributions are a standard for discrete action spaces, while Gaussian distributions are preferred for continuous actions – another major advantage over action-value methods. Here, the stochasticity is induced by sampling actions over action preferences. This ability is key in decision problems where the optimal policies are inherently stochastic, such as in POMDPs. It is also key for credit assignment, since it allows representing equiprobable actions when the advantage is always zero in specific states.

2.6.3.1 Actor-critic methods

Until now, we described policy gradient methods where the return target was a real observed outcome. When the return target is a learned estimate of the full Monte-Carlo return, policy gradient methods take the name of *actor-critic methods*: Actor-critic agents learn a dual model, one for the policy – the *actor* – and one for the value function – the *critic* – that symbiotically interact.

$$\theta_{i+1} = \theta_i + \alpha \left[Q(s, \pi(s)) \gamma \nabla^{\theta} \log \pi(a|s, \theta) \right] \tag{2.36}$$

where $Q(s, \pi(s))$ is an approximation of the true return $G(s)$. The properties of an actor-critic implementation inherit from action-value methods, and mostly depend on the properties of the critic. For example, a biased critic, such as a one-step q-function produces a biased actor-critic algorithm, whereas a full Monte-Carlo return translates into policy learning with high variance. One interpretation of this architecture is that the actor learns to improve the policy, based on the estimated expected returns – the critic. The critic, on the other hand, uses temporal difference to approximate the value function of the actor. This recursive dialogue allows to the critic to “critique“ the actor’s preferences by communicating temporal difference errors. Another interpretation of actor-critic algorithms comes from neuroscience. Speculations about the biophysical significance of actor-critic have driven experimenters to investigate whether the brain implements such dialectic. Preliminary evidence (O’Doherty et al., 2004; Takahashi et al., 2008) suggests that two structures in the brain may function one like an actor and the other as critic: respectively, the dorsolateral and ventral striatum. Dopamine targets both areas, and modulates synaptic plasticity differently in each of those, according to the function of the afferent structure.

2.7 Deep RL

In simple settings, the policy can be represented as a look-up table – with states as keys and actions as values – or a linear combination of features. However, when observations are high dimensional or actions spaces are large, these policies struggle to perform. On one hand, the memory required by tables is too expensive even for the contemporary infrastructures, and tabular method do not promise to scale up. On the other hand, linear representations are not expressive enough, and often fail to encompass the information required to act optimally, or require strong learning biases that diminish the autonomy of the agents. With the reinstatement of dominance of *learning* over symbolic methods, *Deep RL* scales up canonical RL methods to high dimensional problems. The combination of the non-linear behaviour of *Deep Neural Networks*, gradient descent optimisation, and the strong learning biases of RL allowed solving always more complex decision problems. Many of these environments shift the problem to raw pixel observations, introducing the challenge to learn to perceive other than only learn to act optimally.

The main interpretation of a deep RL architecture, and the one we assume in this dissertation, is to view the network as two connected parts: *(i)* a non-linear representation function $z = \phi(o)$, and *(ii)* a linear controller with parameters θ. The final preferences $\pi(a|s, \theta, \phi)$ write as follows:

$$\pi(a|s, \theta, \phi) = \phi(o)^{\intercal}\theta, \quad \forall o \in O \tag{2.37}$$

Alongside sample efficiency, and the non-stationarity induced by the continuously changing features, the problem that most affects credit assignment in these settings is the tendency of the policy to overfit to one specific optimal policy. The low sample efficiency of deep RL requires focusing on a small subset of behaviour. Often, some of these actions are not necessary to obtain a reward, and the agent lacks the understanding that only a subset of its actions affect returns.

2.8 Summary and conclusions

This chapter introduced the fundamental concepts of RL to understand the rest of the manuscript. We started by defining the problem of decision making, and then introduced the Markov Decision Process as a formalism to describe it. We then described the optimal policy problem, and the main methods to solve it. We introduced the GPI schema, and then branched out to two main classes of algorithms: action-value methods and policy gradient methods. Finally, we described how these adapt to approximate settings where knowledge acquisition is limited, and briefly illustrated the notion of Deep RL. The next chapter introduces the set of fundamental assumptions that underpin the rest of the dissertation.

Chapter 3

Influences

In this section, we describe the fundamental influences that shaped the field of Reinforcement Learning today, and the assumptions that they bring to the table. We describe the philosophical biases that underpin the field, the mathematical tools that are used to model decision-making, the psychological theories that inspired the first learning algorithms, and the computational power that made the field possible. Each of these influences will yield a **fundamental assumption**, which we will use to build on the theory of Reinforcement Learning in the following chapters.

The set of influences is inherently multidisciplinary. Often, the very same book (e.g., Pavlov's *"Conditioned Reflexes"* (Pavlov, 1927)) is turned inside-out by separate disciplines at once (e.g., psychologists, neuroscientists, physicists, economists, managers). It is not a fortuity that Turing's 1950 seminal paper on thinking machines (Turing, 1950) was published in a philosophy journal, *Mind*. There exist several versions of this story (Buchanan, 2005; Haenlein & Kaplan, 2019; Sutton & Barto, 2018; Smiley, 2010), depending on their focus and their interpretation of controversies (e.g., when was the first computer invented?).

Here, we are interested in the part of the story that brought conspicuous innovations to defining Reinforcement Learning today. This is by no means a comprehensive story, leaving aside many prominent events and actors, but only a very brief overview.

We recognise four main threads, each corresponding to four major areas of research today: (*i*) the neuropsychologists' research on *conditioning*, and the cognitive sciences concerned with understanding how information in the brain is acquired, stored, processed and retrieved; (*ii*) the physics of optimal control; (*iii*) the philosophical assumptions at the heart of behavioural studies; and finally, (*iv*) computers.

3.1 Neuropsychology

Behavioural psychology and Thorndike's connectionist theory pioneered two main ideas: trial-and-error learning, and classical conditioning theory. The *law of effect* (Thorndike, 1898) introduced the paradigm of stimulus-response (S-R) pairing for learning: when a stimulus eliciting a response is followed by satisfaction (or dissatisfaction), the probability of that very response, at the occurrence of that very stimulus, increases (decreases). There exist two major theories of conditioning. First, *classical* conditioning received its name with Pavlov (Pavlov, 1927), who advanced and detailed Thornedike's work on reflexes. Second, Skinner's *operant* conditioning (Skinner, 1963) is the hallmark of learning by reinforcements. While in the Pavlovian experiments a reinforcer is paired with a stimulus, in *operant* behaviour it is contingent upon a response. In other words, the *operant* - the elicited behaviour - *anticipates* the *reinforcement* - the event that strengthens behaviour[1]. The importance of this subtle difference is paramount. While stimulus-response pairing is a condition for *reflex*, operant behaviour is an attempt to act, to increase (decrease) one's own satisfaction (dissatisfaction). For short, in classical conditioning rewards and punishments are independent of the actions. Instead, in operant

[1]Rewards *are* reinforcements.

conditioning actions determine what reinforcement – hence what reward – is provided. For example, imagine to smell the aroma of your favourite food. The immediate joy, and the immediate excitement is a typical reflex explained by classical conditioning. The event – the smell – produces reward – the joy – but does not require any actions: just the neural acknowledgement of the sensory information. On the other hand, consider an employer offering a fat bonus to their employees because they reached a target. The reward – the bonus – reinforces the employees' behaviour – reaching a target. In this case the reward is contingent upon the action "working smart to reaching the target".

In Neuroscience, Hebbian theory (Hebb, 1949) harmonised the biological functions of the brain for learning with the higher level functioning of the mind. Models of Hebbian learning are the first example of *Artificial Neural Network*. At their foundation there is learning through Hebb's law. Hebb conjectured that if the activation of a neuron A often follows the one of neuron B, the synapse between the two is strengthened. This is often popularly paraphrased as "*neurons that fire together, wire together*". Artificial neural networks are now an indispensable component of the modern Reinforcement Learning formulation, and capable to mimic (in principle) a substrate of biological neurotransmission in order to compose high level abstractions.

Much later, Dayan, Schultz, Montague and Sejnowsky discovered profound connections between the dopaminergic metabolism in the ventral-tegmental area of the brain and reinforcement learning. Experimental evidence in monkeys (Schultz, 1998) suggests that the phasic activity of dopaminergic neurons conveys a peculiar error signal: the difference between an estimate of the expectation of satisfaction, and its true, realised outcome of satisfaction. This hypothesis is regarded as the "*reward prediction error hypothesis of dopamine neuron activity*" (Montague et al., 1996; Schultz et al., 1997). In other words, when a reward is expected, but not collected – or vice versa – dopamine provides a prediction error signal for reward: *surprise*.

From the influences of neuropsychology, we draw the following assumptions, which we also summarise in **Figure 3.1b**.

Assumption 1 (Primary rewards.). There exists a **primary reward** signal that the agent can sense and that is contingent upon the agent's actions, and the state of the world. From the agent's point of view, the primary reward is a function of the *future*, that is, the agent's actions determine the reward. The primary reward is external, innate, and independent from the agent.

Assumption 2 (Secondary reward.). There exists a **secondary reward** signal, which is instead learnt from experience, and depends on the agent. The secondary reward *actualises* the primary reward, transporting it from the future to the present. We refer to *credit* as the quantity that measures this reward.

3.2 Optimal control theory

Despite the several points of contact between psychologists and neuroscientists, the theory of *optimal control* developed an independent, parallel thread. The term *optimal control theory* refers to the mathematical framework concerned with finding the configuration of a dynamic system that minimises a certain cost. Here a *controller* can intervene on a dynamic system, which is formulated as a random process. We owe to Pontryagin (Boltyansky et al., 1956) the bases of this field, who advanced the groundwork of Hamilton and Jacobi in Hamiltonian mechanics. Its main contribution is his *maximum principle* (Pontryagin et al., 1962), which made high dimensional problems more tractable. Its significance lies in the fact that maximising over the Hamiltonian converts an infinite-dimensional control problem – the function space of the Hamiltonian – to a problem of space-discrete optimisation. Richard Bellman (Bellman, 1957) extended Pontryagin's work to time-discrete problems. The new problem formulation took the name of MDP, and their solution – dubbed

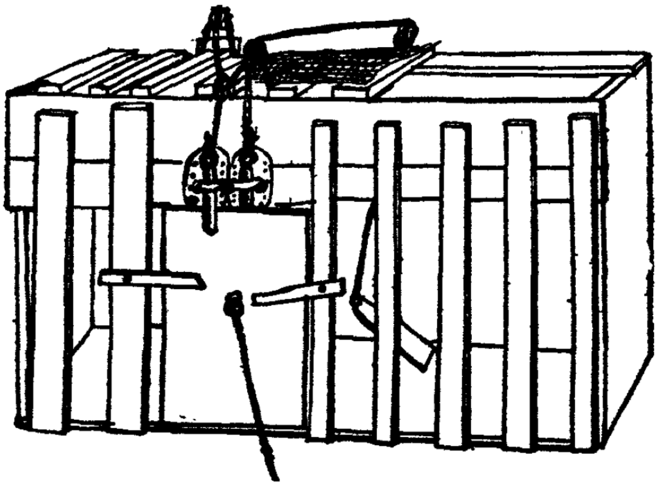
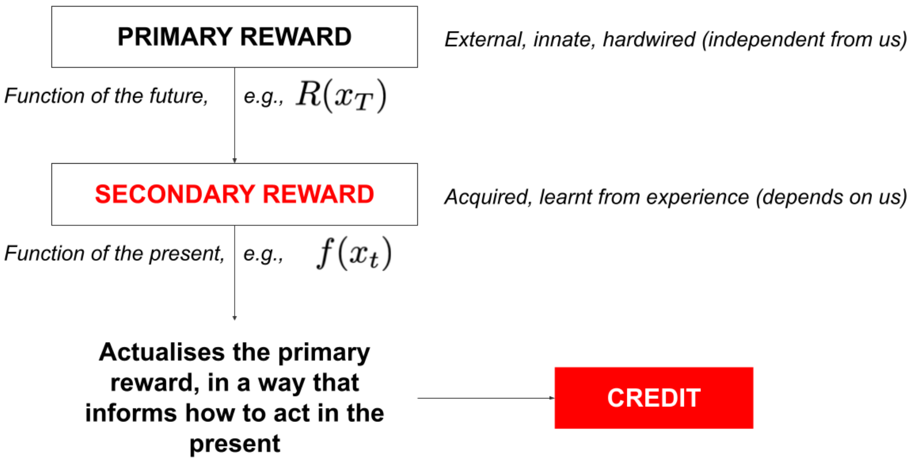


Fig. I.

(a) One of the eleven original Thorndike's cat puzzle box apparatus, used to replace the coeval anecdotal reports on behaviour with experimental observations. The door of the box is held in place by a string anchored to a lever. Each of the eleven versions of the box provide different mechanisms to open the door. The image is adapted from the original publication (Thorndike, 1898).



(b) Primary and secondary rewards in RL and their relationship. The primary reward is external, innate, and independent from the agent. The secondary reward is learnt from experience, and depends on the agent.

dynamic programming – can be derived from the equation that later took Bellman's name. Even today, Markov Decision Processes, the Bellman equation, and dynamic programming are the workhorse of many Reinforcement Learning formulations.

Assumption 3 (Optimal control.). From optimal control theory, we assume that any decision-making problem can be formulated as a MDP, and that the optimal policy can be derived from the Bellman equation.

Section 2 further elaborates on the topic.

3.3 Philosophy

With time, the field developed an independent identity, and nowadays, the existence of a distinct science of human decision-making is an established fact. The research has set the shared goal of understanding how preferences are formed and their causes, and draws on many assumptions from the underlying disciplines, despite often not explicitly (Andersen et al., 2019). One key philosophical bias is that behaviour is lawful and determined (Skinner, 1953). In behavioural studies that predate the 1900, the prevailing dualist and compatibilist philosophy (Clark, 1999) regarded behaviour as the product not of specifiable antecedent conditions, but of an inner "will": a causally privileged device empowered to interfere with the naturalistic stream of events by means of freedom[2]. Pity the natural compatibilists, for this is a major philosophical bias that affects modern decision-making theory, allowing science to claim general laws from experimental results. Experimenters can expect that a decision is the result of specificable conditions, and if these factors are identified, behaviour can be univocally predicted, and, to some extent, controlled. The

[2]The compatibilist view is not abandoned today, and debate is ongoing (Carroll, 2012). However, the discussion has philosophy at its centre. Even if the reach of neuroscience, physics, and behavioural psychology studies towards philosophy is strong, the practical impacts of philosophy on those, are weak.

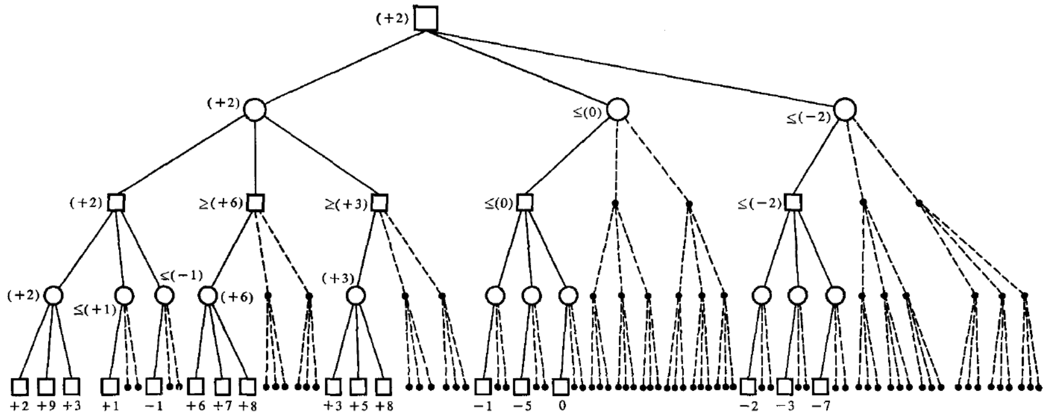


Figure 3.2: The "look-ahead" move tree originally presented by Samuel (Samuel, 1959) in his preliminary studies on using machine learning to play checkers. The squares show the look-ahead move for the first player, whereas circles represent the board positions for the opponent. Dashed lines are branches removed from search. The term "look-ahead" is now widely used in reinforcement learning.

premise of rationality (Samuelson, 1937) underpins this idea. The rational decision maker has no limit to the amount of knowledge it may acquire, and it is capable of ordering its preferences when complete information is available. Alas, after 1950 scholars have gradually come to terms with the inability of humans to make perfect, rational decisions, and a long history of how to make decisions in the face of uncertainty has begun (Simon, 1957).

Assumption 4 (Naturalism and rationality.). From the naturalism and rationality, we assume that behaviour is lawful and determined. This means that, given a set of conditions, the decision-maker will always make the same decision. For this reason, behaviour is predictable and can be deterministically controlled.

3.4 Computers

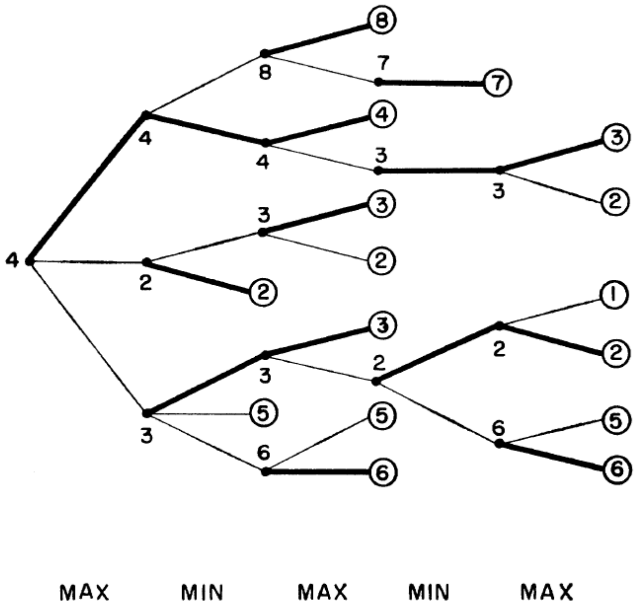


Figure 3.3: The original "back-up" from Minsky (Minsky, 1961) for an arbitrary game tree. The first three lines, from the left, fan out the *first player*'s moves, while the second level of branches the *opponent*'s. To "back-up" a decision tree is the process to evaluate a policy: how much reward will I collect, if I follow a specific set of moves, in expectation to the opponent's moves?

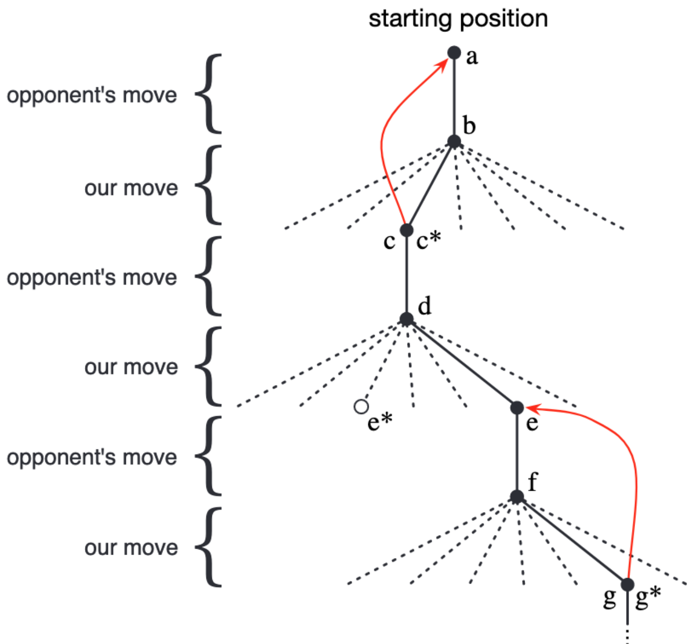


Figure 3.4: Contemporary "back-up" diagram from Sutton (Sutton & Barto, 2018) for the game of Tic-Tac-Toe. The solid black line represents actions that the agent, or the opponent, have taken. Starred moves are optimal moves, and red arrows are "back-ups". The move e was not optimal, but targeted exploration instead, and as such, it is not "backed-up", providing no reinforcement signal.

While these threads of research proceeded with their goals, the era of computers was about to begin. In the first part of the 20th century, computers were mostly specific-purpose machines. The first proposal for a general-purpose computer – and Turing machine – has been Babbage's *analytical engine*, later published by Menabrea and Lovelace (Menabrea & Lovelace, 1843). These computers are often not considered more than clockwork machinery, and of no interest for the purpose of decision-making. When, in the second part of the twentieth century, machines grew the power to replicate real experiments, the quest of understanding the nature of intelligent behaviour began. In 1956, with the Dartmouth Summer Research Project on Artificial Intelligence, Artificial Intelligence officially acquired its name.

Research before 1960 was centred around the idea of heuristic search and how to effectively reduce the breadth and depth of search (Minsky, 1968). Samuel's (Samuel, 1959) checkers program was the first effective use of heuristic search for decision making. *Learning* is the central topic in these years. Samuel (Samuel, 1967) itself introduced some of the concepts that will be systematised much later in the reinforcement learning literature, such as *temporal-difference* learning, and the evaluation of decision sequences by "*backing-up*" a decision, shown in Figures 3.2, 3.3, and 3.4.

The 1960s saw a paradigm shift. The concern became less with *learning*, and more with the problem of knowledge representation, however acquired. In these years decisions are the result of the logical manipulation of symbols, leading to the stagnation of statistical methods.

Finally, the Deep Learning – and the Deep Reinforcement Learning – revolution is likely the last breakthrough in computer learning. In 2012, thanks to the power of GPUs and the ubiquitous availability of data, artificial neural networks made a comeback. AlexNet (Krizhevsky et al., 2012) re-established the dominance of learning methods over the manipulation of symbols, and became the state-of-the art for computer vision. Since then, deep learning has made huge leap forwards. In 2013 deep reinforcement learning was born (Mnih et al., 2013), and in 2015 Deep Q-Networks surpassed human experts in playing Atari video-games (Mnih et al., 2015).

Assumption 5 (Computational decision-making.). In this dissertation, we assume that the decision-making problem can be formulated as a computational problem, and that we can use computers to solve it.

3.5 Summary and conclusions

In this chapter, we have identified the main influences that have shaped the field of Reinforcement Learning. We have identified four main threads of research: neuropsychology, optimal control theory, philosophy, and computer science. From these influences, we have drawn five fundamental assumptions that will guide our development of the theory of Reinforcement Learning in the following chapters.

From the neuropsychology, we have drawn the assumptions that there exists a primary reward signal, and a secondary reward signal. The primary reward is external, innate, and independent from the agent, while the secondary reward is learnt from experience, and depends on the agent, laying the foundation for a measure of credit. From the optimal control theory, we have assumed that any decision-making problem can be formulated as a MDP, and that the optimal policy can be derived from the Bellman equation. From the philosophy, we have assumed that behaviour is predictable and deterministically controllable: given a set of conditions, the decision-maker will always make the same decision. Finally, from computer science, we have assumed to solve the decision-making problem using computers, and in particular, Deep RL.

Part III

Understanding the Credit Assignment Problem

Despite its central role, there is little discussion on the precise mathematical nature of credit. While these proxies are sufficient to unlock solutions to complex tasks, it remains unclear where to draw the line between a generic measure of action influence and *credit*. Existing works focus on partial aspects or sub-problems (Hung et al., 2019; Arjona-Medina et al., 2019; Arumugam et al., 2021) and not all works refer to the CAP explicitly in their text (Andrychowicz et al., 2017; Nota et al., 2021; Goyal et al., 2019a), despite their findings providing relevant contributions to address the problem. The resulting literature is fragmented and lacks a space to connect recent works and put their efforts in perspective for the future.

In this Part, we aim to answer the **Question 1** defined in **Chapter 1.5.1**, by proposing a formalism to define credit and the CAP.

We split **Question 1** in three sub-questions:

Question 1.1. *What* is the *credit* of an action? How is it different from an *action value*? And what is the CAP? What in words, and what in mathematics?

Question 1.2. *How* do agents learn to *assign* credit? What are the main methods in the literature and how can they be organised?

Question 1.3. How can we *evaluate* whether a method is improving on a challenge? How can we monitor advancements?

Chapter 4

Quantifying action influences

We start by answering **Question 1.1.**, which aims to address the problem of *what* to measure, when referring to credit. Since Minsky (1961) raised the Credit Assignment Problem (CAP), a multitude of works paraphrased his words:

- "*The problem of how to incorporate knowledge*" and "*given an outcome, how relevant were past decisions?*" (Harutyunyan et al., 2019),
- "*Is concerned with identifying the contribution of past actions on observed future outcomes*" (Arumugam et al., 2021),
- "*The problem of measuring an action's influence on future rewards*" (Mesnard et al., 2021),
- "*An agent must assign credit or blame for the rewards it obtains to past states and actions*" (Chelu et al., 2022),
- "*The challenge of matching observed outcomes in the future to decisions made in the past*" (Venuto et al., 2022),
- "*Given an observed outcome, how much did previous actions contribute to its realization?*" (Ferret, 2022, Chapter 4.1).

These descriptions converge to Minsky's original question and show agreement

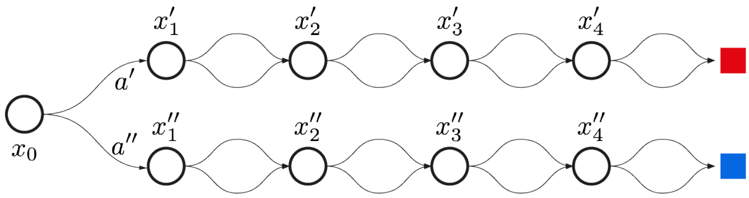


Figure 4.1: A simplified MDP to develop an intuition of credit. The agent starts in x_0, and can choose between two actions, a' and a'' in each state; the reward is 1 when reaching the upper, solid red square, and 0 otherwise. The first action determines the outcome alone.

in the literature on an informal notion of credit. In this chapter, we propose to reflect on the different metrics that exist in the literature to quantify it. We generalise the idea of *action value*, which often only refers to q-values, to that of *action influence*, which describes a broader range of metrics used to quantify the credit of an action. While we do not provide a definitive answer on what credit *should* be, we review how different works in the existing RL literature have characterised it. We now start by developing an intuition of the notion of credit.

Consider Figure 4.1, inspired to both Figure 1 of Harutyunyan et al. (2019) and to the *umbrella* problem in Osband et al. (2020). The action taken at x_0 determines the return of the episode by itself. From the point of view of *control*, any policy that always takes a' in x_0 (i.e., $\pi^* \in \Pi^* : \pi^*(a'|x_0) = 1$), and then any other action afterwards, is an optimal policy. From the CAP point of view, some optimal actions, namely those after the first one, do not *actually* contribute to optimal returns. Indeed, alternative actions still produce optimal returns and contribute equally to each other to achieve the goal, so their credit is equal. We can see that, in addition to optimality, credit not only identifies optimal actions but informs them of how *necessary* they are to achieve an outcome of interest.

From the example, we can deduce that credit evaluates actions for their potential to influence an outcome. The resulting CAP is the problem of **estimating the influence** of an action over an outcome from experimental data and describes a pure association between them.

Why solving the CAP? Action evaluation is a cornerstone of RL. In fact, solving a control problem often involves running a GPI scheme. Here, the influence of an action drives learning, for it suggests a possible direction to improve the policy. For example, the action-value plays that role in Equation (2.22). It follows that the quality of the measure of influence fundamentally impacts the quality of the policy improvement. Low quality evaluations can lead the policy to diverge from the optimal one, hinder learning, and slow down progress (Sutton & Barto, 2018; van Hasselt et al., 2018). On the contrary, high quality evaluations provide accurate, robust and reliable signals that foster convergence, sample-efficiency and low variance. While simple evaluations are enough for specialised experiments, the real world is a complex blend of multiple, sometimes hierarchical tasks. In these cases, the optimal value changes from one task to another, and these simple evaluations do not bode well to adapt to general problem solving. Yet, the causal structure that underlies the real word is shared among all tasks, and the modularity of its causal mechanisms is often a valuable property to incorporate. In these conditions, learning to assign credit in one environment becomes a lever to assign credit in another (Ferret et al., 2021a), and ultimately makes learning faster, more accurate and more efficient. For these reasons, and because an optimal policy only requires discovering one single optimal trajectory, credit stores knowledge beyond that expressed by optimal behaviours alone, and solving the control problem is not sufficient to solve the CAP, with the former being an underspecification of the latter.

4.1 Are all *action values, credit*?

As we stated earlier, most Deep RL algorithms use some form of *action influence* to evaluate the impacts of an action on an outcome. This is a fundamental requirement to rank actions and select the optimal one to solve complex tasks. For example, many model-free methods use the *state-action value* function $q^\pi(s, a)$ to evaluate actions (Mnih et al., 2015; van Hasselt et al., 2016), where actions contribute as much as the expected return they achieve at termination of the episode. Advantage Learning (AL) (Baird, 1999; Mnih et al., 2016; Wang et al., 2016b, Chapter 5) uses the *advantage* function $A^\pi(s_t, a_t) = q^\pi(s_t, a_t) - v^\pi(s_t)$ [1] to measure credit, while other works study the effects of the *action-gap* (Farahmand, 2011; Bellemare et al., 2016; Vieillard et al., 2020b) on it, that is, the relative difference between the expected return of the best action and that of another action, usually the second best. Action influence is also a key ingredient of actor-critic and policy gradient methods (Lillicrap et al., 2015; Mnih et al., 2016; Wang et al., 2016a), where the policy gradient is proportional to $\mathbb{E}_{\mu,\pi}[A^\pi(s, a)\nabla \log \pi(A|s)]$, with $A^\pi(s, a)$ estimating the influence of the action A.

These proxies are sufficient to select optimal actions and unlock solutions to complex tasks (Silver et al., 2018; Wang et al., 2016b; Kapturowski et al., 2019; Badia et al., 2020; Ferret et al., 2021b). However, while many works explicitly refer to the action influence as a measure of credit, the term is not formally defined and, it remains unclear where to draw the line between *credit* and other quantities. Key questions arise: *What is the difference between these quantities and credit? Do they actually represent credit as originally formulated by Minsky (1961)? If so, under what conditions do they do?* Without a clear definition of *what* to measure, we do not have an appropriate quantity to target when designing an algorithm to solve the CAP. More importantly, we do not have an appropriate quantity to use as a single source of truth and term of reference to measure the accuracy of other metrics of action influence, and how well they approximate credit. To fill this gap, we proceed as follows:

- Section 4.2 formalises what is a *goal* or an *outcome*: what we evaluate

[1]To be consistent with the RL literature we abuse notation and denote the advantage with a capital letter A^π despite not being random and being the same symbol of the action A_t.

the action for;

- Section 4.3 unifies existing functions under a common formalism;
- Section 4.4 formalises the CAP following this definition;
- Section 4.6 analyses how different works interpreted and quantified *action influences* and reviews them;
- Section 4.7 distils the properties that existing measures of action influence exhibit.

We suggest the reader only interested in the final formalism to directly skip to Section 4.4, and to come back to the next sections to understand the motivation behind it.

4.2 What is a *goal*?

Because credit measures the influence of an action upon achieving a certain goal, to define credit formally we must be able to describe *goals* formally, and without a clear understanding of what constitutes one, an agent cannot construct a learning signal to evaluate its actions. *Goal* is a synonym for *purpose*, which we can informally describe as a performance to meet or a prescription to follow. Defining a goal rigorously allows making the relationship between the action and the goal explicit (Ferret, 2022, Chapter 4) and enables the agent to decompose complex behaviour into elementary ones in a compositional (Sutton et al., 1999; Bacon et al., 2017), and possibly hierarchical way (Flet-Berliac, 2019; Pateria et al., 2021; Hafner et al., 2022). This idea is at the foundation of many CA methods (Sutton et al., 1999; 2011; Schaul et al., 2015a; Andrychowicz et al., 2017; Harutyunyan et al., 2019; Bacon et al., 2017; Smith et al., 2018; Riemer et al., 2018; Bagaria & Konidaris, 2019; Harutyunyan et al., 2018; Klissarov & Precup, 2021). We proceed with a formal definition of *goals* in the next paragraph, and review how these goals are *represented* in seminal works on CA in the one after. This will lay the foundation for a unifying notion of credit later in Sections 4.3.

Defining goals. To define goals formally, we adopt the *reward hypothesis*, which posits:

> *That all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward).* (Sutton, 2004).

Here, the goal is defined as the *behaviour* that results from the process of maximising the return. The reward hypothesis has been further advanced by later studies (Abel et al., 2021b; Pitis, 2019; Shakerinava & Ravanbakhsh, 2022; Bowling et al., 2023). In the following text, we employ the goal definition in Bowling et al. (2023), which we report hereafter:

Definition 1 (Goal)**.** *Given a distribution $\mathbb{P}(H)$ over a set of finite histories $\mathcal{H}$ with $H \in \mathcal{H}$, we define a* goal *as a partial ordering over elements of $\mathcal{H}$, and for all $h, h' \in \mathcal{H}$ we write $h \succsim h'$ to indicate that h is preferred to h' or that the two are indifferently preferred.*

Here, H is a random history in the set of all histories $\mathcal{H}$ as described in **Chapter 2**, and $\mathbb{P}(H)$ is an unknown distribution over histories, different from that induced by the policy and the environment. An agent behaviour and an environment then induce a new distribution over histories, and we obtain $\mathbb{P}_{\mu,\pi}(H)$ as described in **Chapter 2**. This in turn allows defining a partial ordering over policies, rather than histories, and we write analogously $\pi \succsim \pi'$ to indicate the preference. For the *Markov Reward Theorem* (Bowling et al., 2023, Theorem 4.1) and under mild conditions (Bowling et al., 2023), there exists a deterministic, Markov reward function[2] $R : \mathcal{O} \times \mathcal{A} \to [0, 1]$ such

[2]We omit the transition dependent discounting for the sake of conciseness and because not relevant to our problem. The reader can consult Pitis (2019); White (2017) for details.

that the maximisation of the expected sum of rewards is consistent with the preference relation over policies.

Subjective and objective goals. The *Markov Reward Theorem* holds both if the preferences are defined *internally* by the agent itself – this is the case of *intrinsic motivation* (Piaget et al., 1952; Chentanez et al., 2004; Barto et al., 2004; Singh et al., 2009; Barto, 2013; Colas et al., 2022) – and in case they originate from an *external* entity, such as an agent-designer. In the first case, the agent doing the maximisation is the same as the one holding the ordering over policies, and we refer to the corresponding goal as a *subjective goal.* In the second case, an *agent designer* or an unknown, non-observable entity holds the ordering and a separate *learning agent* is the one pursuing the optimisation process. We refer to a goal as an *objective goal* in this latter case. These settings usually correspond to the distinction between goals and sub-goals in the literature (Liu et al., 2022).

Outcomes. A particularly interesting use of goals for CA is in hindsight (Andrychowicz et al., 2017). Here the agent acts with a goal in mind, but it evaluates a trajectory as if *a* reward function – one different from the original one – was maximised in the current trajectory. We discuss the benefits of these methods in Section 6.4. When this is the case, we use the term *outcome* to indicate a realised goal in hindsight. In particular, given a history $H \sim \mathbb{P}_{\mu,\pi}(H)$, there exists a deterministic, Markov reward function R that is maximal in H. We refer to the corresponding H as an outcome. For example, consider a trajectory h that ends in a certain state s. There exist a Markov reward function that outputs always 0 and 1 only when the s is the final state of h. We refer to h as an *outcome.*

In other words, this way of defining goals or outcomes corresponds to defining a task to solve, which in turn can be expressed through a reward function with the characteristics described above. Vice-versa, the reward function can *encode* a task. When credit is assigned with respect to a particular goal or outcome, it then evaluates the influence of an action to solving that particular task. As discussed above, this is key to decomposing and recomposing complex behaviours and the definition aligns with that of other disciplines, such as psychology where *a goal ... is a cognitive representation of something that is possible in the future* (Elliot & Fryer, 2008) or philosophy, where representations do not merely read the world as it is, but they express *preferences* over something that is possible in the future (Hoffman, 2016; Prakash et al., 2021; Le Lan et al., 2022).

Representing goals and outcomes. Expressing the relation between actions and goals explicitly, that is, when the function that returns the credit of an action has a goal as an input, raises the problem of how to *represent* a goal for computational purposes. This is important because among the CA methods that define goals explicitly (Sutton et al., 2011; Schaul et al., 2015a; Andrychowicz et al., 2017; Rauber et al., 2019; Harutyunyan et al., 2019; Tang & Kucukelbir, 2021; Arulkumaran et al., 2022; Chen et al., 2021), not many of them use the rigour of a general-purpose definition of goal such as that in Bowling et al. (2023). In these works, the *goal-representation space*, which we denote as $g \in \mathcal{G}$, is arbitrarily chosen to represent specific features of a trajectory. It denotes an *object*, rather than a performance or a prescription to meet. For example, a *goal-representation* g can be a state (Sutton et al., 2011; Andrychowicz et al., 2017) and $g \in \mathcal{G} = \mathcal{S}$. It can be a specific observation (Nair et al., 2018) with $g \in \mathcal{G} = \mathcal{O}$. Alternatively, it can be an abstract features vector (Mesnard et al., 2021) that reports on some characteristics of a history, and we have $g \in \mathcal{G} = \mathbb{R}^d$, where d is the dimensionality of the vector. Even, a goal can be represented by a natural language instruction (Luketina et al., 2019) and $g \in \mathcal{G} = \mathbb{R}^d$ is the embedding of that piece of text. A goal can be represented by a scalar $g \in \mathcal{G} = \mathbb{R}$ (Chen et al., 2021) that indicates a specific return to achieve, or even a full command (Schmidhuber, 2019), that is a return to achieve is a specific window of time.

While these representations are all useful heuristics, they lack formal rigour and leave space for ambiguities. For example, saying that the goal is a state might mean that *visiting the state at the end of the trajectory* is the goal or that visiting it in the *middle* of it is the goal. This is often not formally defined, and what is the reward function that corresponds to that specific representation of a goal is not always clear. In the following text, when surveying a method or a metric that specifies a goal, we refer to the specific goal representation used in the work and make an effort to detail what is the reward function that underpins that goal representation.

4.3 The *influence function*

Having established a formalism for goals and outcomes, we are now ready to describe *credit* formally and we proceed with a formalism that unifies the existing measures of action influence. We first describe a generic function that generalises most CAs, and then proceed to formalise the CAP. Overall, this formulation provides a term of reference for the quantities described in Section 4.6. We now formalise an *influence function*:

Definition 2 (Influence function)**.** *Consider an action $a \in \mathcal{A}$, a goal $g \in \mathcal{G}$, and a context $c \in \mathcal{C}$. We use the term* influence function *to denote a function $\mathcal{K}$ that maps a context, an action, and an outcome to a quantity $y \in \mathcal{Y}$, which we refer to as the* ***influence*** *of the action:*

$$K : \mathcal{C} \times \mathcal{A} \times \mathcal{G} \rightarrow \mathcal{Y}. \tag{4.1}$$

Here, a context $c \in \mathcal{C}$ represents some input data and can be arbitrarily chosen depending on the assignment in question. For example, c can be a state s. A context must hold information about the present, for example, the current state or the current observation; it may contain information about the past, for example, the sequence of past decisions that occurred until now for a POMDP; to evaluate the current action, it can contain information about what future actions will be taken *in-potentia*, for example by specifying a policy to follow when $a \in \mathcal{A}$ is not taken, or a fixed trajectory, in which case the current action is evaluated in hindsight (Andrychowicz et al., 2017).

In the general case, the action influence is a random variable $Y \in \mathcal{Y} \subset \mathbb{R}^d$. This is the case, for example, of the action-value distribution (Bellemare et al., 2017) as described in Equation 4.7, where the action influence is defined over the full distribution of returns. However, most methods extract some scalar measures of the full influence distribution, such as expectations (Watkins, 1989), and the action influence becomes a scalar $y \in \mathbb{R}$. In the following text, we mostly consider scalar forms of the influence $\mathcal{Y} = \mathbb{R}$ as these represent the majority of the existing formulations.

In practice, an *influence function* provides a single mathematical form to talk about the multitude of ways to quantify action influence that are used in the literature. Influence functions work in *forethought*. They take an action $a \in \mathcal{A}$, the current context $c \in \mathcal{C}$ and a goal $g \in \mathcal{G}$ and map them to some measure of *action influence* that expresses information about the *future*. While maintaining the same mathematical form, different assignments can return different values of action influence and steer the improvement in different directions.

Equation (4.1) also resembles the General Value Function (GVF) (Sutton et al., 2011), where the influence $y = q^{\pi}(s, a, g)$ is the expected return of the policy π when taking action a in state s, with respect a goal g. However, in GVFs: *(i)* y is an *action value* and a scalar, and does not generalise other forms of action influence; *(ii)* the goal is only an MDP state $g \in \mathcal{S}$ and does not generalise to our notion of goals in Section 4.2; *(iii)* the context is only the current state and does not contain information about the future, as it happens, for example, with hindsight (Andrychowicz et al., 2017). Table 4.1 at page 99 contains further details on the comparison and further specifies the relationship between the most common functions and their corresponding assignment.

4.4 The *assignment*

The generality of the assignment formalism reflects the great heterogeneity of action influence metrics, which we review later in Section 4.6. This heterogeneity shows that, even if most studies agree on an intuitive notion of credit, they diverge in practice on how to quantify credit mathematically. Having unified existing influence functions in the previous section, we now proceed to formalise the CAP. This allows us to put the existing methods into a coherent perspective as a guarantee for a fair comparison, and to maintain the heterogeneity of the existing measures of action influence.

Assumptions. We assume standard model-free, Deep RL settings and consider an influence function represented as a neural network $K^\theta : \mathcal{C} \times \mathcal{A} \times \mathcal{G} \times \Theta \to \mathbb{R}$ with parameters $\theta \in \Theta = \mathbb{R}^n$, with $n \ll |\mathcal{S}| \times |\mathcal{A}|$, that can be used to approximate the influence of the actions. This usually represents the critic of an RL algorithm.

We further assume that the agent has access to a set of experiences $\mathcal{D}$ and that it can sample from it according to a distribution $D \sim \mathbb{P}_D$. This can be a pre-compiled set of external demonstrations, and $\mathbb{P}_D = \mathcal{U}$ sampling uniformly over it, or a POMDP, where $\mathbb{P}_D = \mathbb{P}_{\mu,\pi}$, or even a fictitious model of a POMDP $\mathbb{P}_D = \mathbb{P}_{\widetilde{\mu},\pi}$, where $\widetilde{\mu}$ is a function internal to the agent, of the same form of μ. These are mild assumptions as they correspond to, respectively, offline settings, online settings, and model-based settings where the model is learned. We detail these settings in Appendix A.2.

We now define an *assignment* formally.

Definition 3 (Assignment)**.** *We use the term* assignment *to refer to an update rule $\Lambda : \mathcal{K} \times \mathcal{D} \to \mathcal{K}$ that takes an influence function K^θ and some experience* *data* $\mathcal{D} = (\mathcal{S} \times \mathcal{A} \times \mathcal{R})^T$ *to return a new estimate of the assignment function* $K^{\theta'}$.

The assignment effectively generalises the notion of *backup* (Sutton & Barto, 2018), which only considers *temporal* updates, to a more general form of update that can consider any form knowledge acquisition from experience, for example, sharing the update to states that differ from the one visited in features that are not relevant to the current task (Schaul et al., 2015a).

In practice, in Deep RL, assignments apply Λ only to a small subset of the experience data $\{d_T\} \sim \mathbb{P}_D$ because environments are too complex to consider the full, exhaustive set $\mathcal{D}$. Then, an assignment outputs the influence y for all the actions a_t in each trajectory d_T, and Λ takes the form $\Lambda : \mathcal{K} \times \mathcal{D} \to \mathcal{Y}^T$, with often $\mathcal{Y}^T = \mathbb{R}^T$. We refer to the output $y^T \in \mathcal{Y}^T$ as the *target* of the assignment and use the term to *assign credit* to denote the application of Λ to the data $\{d_T\}$ to get $\mathcal{Y}^T$.

Notice that this formulation of an *assignment* is recursive. While this might now always be the case, this is a mild assumption in Deep RL, since the vast majority of assignments are formulated recursively This allows us to generalise the notion of credit assignment to any form of knowledge acquisition from a pair of experience and prior believes.

For example, consider TD learning with 3-step returns. Here, K^θ is the action-value function $q^{\pi,g}(s, a, \theta)$. After experiencing 3 transitions by sampling $d_3 \sim \mathbb{P}_{\mu,\pi}$, our assignment rule Λ calculates a 3-step return for each state-action $(s_t, a_t) \in d_T$, $y = (z_1, z_2, z_3)$. These values y become the target of the assignment, and Λ updates $q^{\pi,g}(s, a, \theta)$ using the assignment rule $q^{\pi,g}(s_t, a_t) \leftarrow q^{\pi,g}(s_t, a_t) + \alpha[z_t - q^{\pi,g}(s_t, a_t)]$. In the following text, we use the two forms $\Lambda : \mathcal{K} \times \mathcal{D} \to \mathcal{K}$ and $\Lambda : \mathcal{K} \times \mathcal{D} \to \mathcal{Y}^T$ interchangeably, depending on whether we include the function update in the operation.

Unlike influence functions, assignments work in *retrospection*, by grounding an influence function in actual experience, as shown in **Figure 4.2** Starting from a guess of influence, it then compares it with some factual data $d \sim \mathcal{D}$, and adjusts the guess to better match the data. The update rule, and so the target, is a key component of most (if not all) credit assignment formulations in the literature, a main focus of study for improving the sample efficiency of RL algorithms, and a main consideration when tuning reinforcement learning algorithms for specific problems.

4.5 The *Credit Assignment Problem*

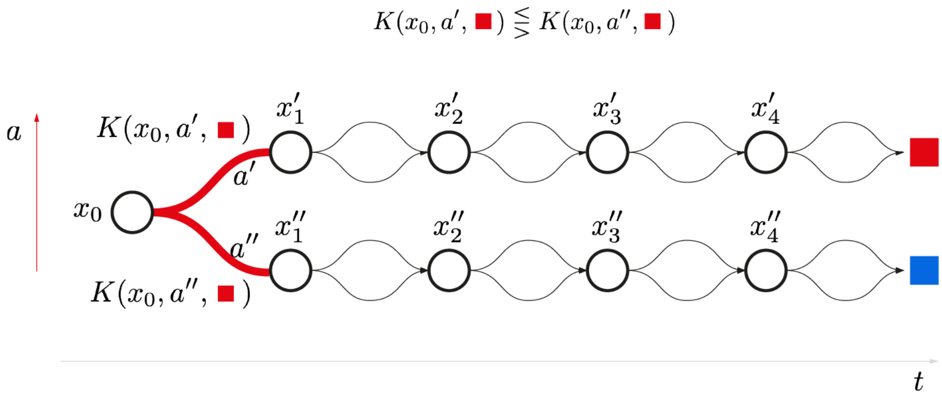
With this notion of assignment, we then cast the CAP as the problem of grounding a measure of action influence in the experience.

Definition 4 (The credit assignment problem)**.** *We refer to the* Credit Assignment Problem *as the problem of finding the set of parameters $\theta \in \Theta$ such that, repeatedly applying the operator Λ to K^θ returns the true influence function K:*

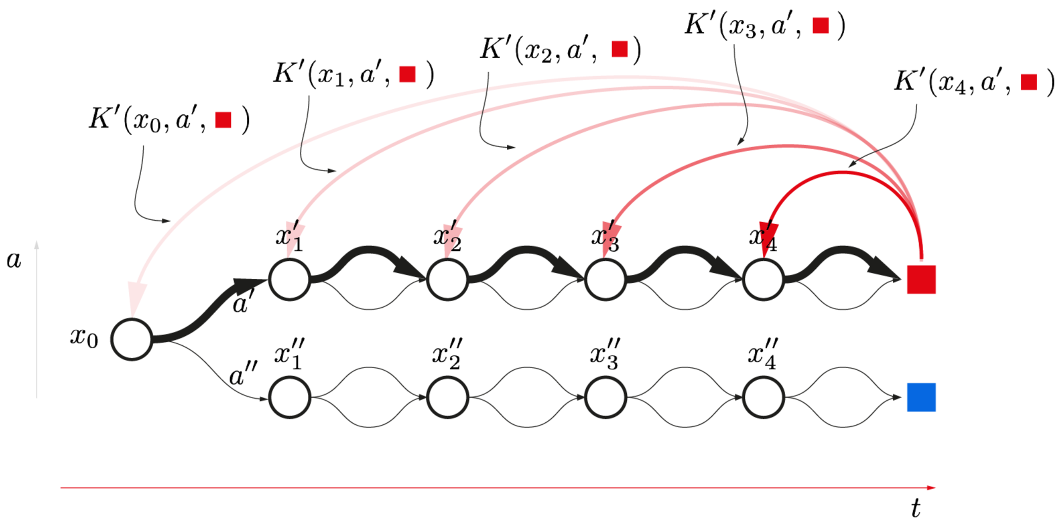
$$\Lambda[K^\theta, \mathcal{D}] = K. \tag{4.2}$$

Notice that, once a particular choice of influence function is made – e.g., a q-function, an advantage function (see Section 4.6 for more) – its value is uniquely determined by its context, the goal and the environment's trajectory distribution. Because this value is grounded in experience – i.e., can be calculated exactly – we refer to it as the *true influence* K. The CAP is then the problem of finding a set of parameters θ such that the probabilistic estimates of the influence function K^θ are exact with respect to the true influence K.

Each CA method specifies its own operator Λ. Despite many works in the



(a) Diagram showing the mechanisms at the base of the influence function in a simple MDP with only two actions.



(b) Diagram showing the mechanisms at the base of the *assignment* in the same MDP. Red arrows represent assignments.

Figure 4.2: Difference between the *influence function* and the *assignment*, as described in Definitions 2 and 3. The *influence function* works in *forethought*. It applies a *forward* perspective to the problem: the influence function is a function of the *present* and outputs a prediction of the *future*. This corresponds to a *prediction* problem. On the contrary, the *assignment* works in *retrospection*. It applies a *backward* perspective to the problem: the assignment is a function of the *future*, with respect to the state that is updated. This corresponds to a *credit assignment* problem. Notice that, while the schema only shows temporal updates for simplicity, the red arrows can point to any state in the environments, according to the CA rule used.

literature of tabular and linear RL have characterised Λ, doing so in Deep RL is more challenging because K^θ is non-linear and the CAP often does not have closed form solution. For this reason, the studies are sparser, for example, van Hasselt et al. (2018). Here, we limit the scope to assume that an operator $\Lambda : \Theta \times \mathcal{D} \to \Theta$ can converge to the true influence K^θ with a certain probability p. Furthermore, we assume that we can calculate empirical statistics of a sequence of updates, such as the empirical variance, convergence rate and fixed-point bias.

Different choices of influence functions have a great impact on the hardness of the problem. In particular, there is a trade-off between: *(a)* how effective the chosen measure of influence is to inform the direction of the policy improvement, *(b)* how easy it is to learn that function from experience, if that is possible at all. For example, using *causal influence* (Janzing et al., 2013) as a measure of action influence makes the CAP hard to solve in practice. This is because discovering causal mechanisms from associations alone is notoriously challenging (Pearl, 2009; Bareinboim et al., 2022), and pure causal relationships are rarely observed in nature (Pearl et al., 2000). However, causal knowledge is reliable, robust to changes in the experience collected and effective, and causal mechanisms can be invariant to changes in the goal.

On the contrary, q-values are easier to learn as they represent a measure of statistical correlation between state-actions and outcomes, but their knowledge is limited to the bare minimum necessary to solve a control problem. This makes them more brittle to sudden changes to the environment, for example, in open-ended settings (Abel et al., 2023).

Which influence function to use for each specific problem, as well as properly characterising the operator Λ in Deep RL, are still the subject of investigation in the literature. Ideally, we should aim for the most effective measure of influence that can be learned with the least amount of experience.

Assignment	Action influence	Context	Action	Goal
State-action-value	$q^{\pi}(s, a)$	$s \in \mathcal{S}$	$a \in \mathcal{A}$	$g \in \mathbb{R}$
Advantage	$q^{\pi}(s, a) - v(s)$	$s \in \mathcal{S}$	$a \in \mathcal{A}$	$g \in \mathbb{R}$
General q-value function	$q^{\pi,R}(s, a)$	$s \in \mathcal{S}$	$a \in \mathcal{A}$	$g \in \mathcal{S}$
Distributional action-value	$Q^{\pi}(s, a)$	$s \in \mathcal{S}$	$a \in \mathcal{A}$	$g \in \{0, \ldots, n\}$
Distributional advantage	$D_{KL}(Q^{\pi}(s, a)\|\|V^{\pi}(s, a))$	$s \in \mathcal{S}$	$a \in \mathcal{A}$	$g \in \{0, \ldots, n\}$
Hindsight advantage	$1 - \frac{\pi(A_t\|s)}{\mathbb{P}_D(A_t\|s_t, Z_t)} Z_t$	$s \in \mathcal{S}, h_T \in \mathcal{H}$	$a \in h$	$g \in \mathbb{R}$
Counterfactual advantage	$\mathbb{P}_D(A_t = a\|S_t = s, F_t = f)q(s, a, f)$	$s \in \mathcal{S}$	$a \in h$	$g \in \mathbb{R}$
Posterior value	$\sum_{t=0}^{T} \mathbb{P}_{\mu,\pi}(U_t = u\|d)v^{\pi}(o_t, x_t)$	$o \in \mathcal{O}, u \in \mathbb{R}^d, \pi$	$A \sim \pi$	$g \in \mathbb{R}$
Policy-conditioned value	$q(s, a, \pi)$	$s \in \mathcal{S}, \pi \in \Pi$	$a \in \mathcal{A}$	$g \in \mathbb{R}$

Table 4.1: A list of the most common *action influences* and their assignment functions in the Deep RL literature analysed in this survey. For each function, the table specifies the influence, the context representation, the action, and the goal representation of the corresponding assignment function $K \in \mathcal{K}$.

4.6 Existing influence functions

We now survey the most relevant influence functions from the literature. The following list is not exhaustive, but rather it is representative of the key properties that an influence function can exhibit. For brevity, and without loss of generality, we omit functions that do not explicitly evaluate actions (for example, state-values), but we notice that it is still possible to reinterpret an assignment to a state as an assignment to a set of actions for it affects all the actions that led to that state.

State-action values (Shannon, 1950; Schultz, 1967; Michie, 1963; Watkins, 1989) are a hallmark of RL, and are described by the following expression:

$$q^{\pi}(s, a) = \mathbb{E}_{\mu,\pi}[Z_t|S_t = s, A_t = a]. \tag{4.3}$$

Here, the context c is a state $s \in \mathcal{S}$ in the case of MDPs or a history $h \in \mathcal{H}$ for a POMDP. The q-function quantifies the credit of an action by the expected return of the action in the context.

Among the simplest ways to quantify credit and offer a basic mechanism to solve control problems there are q-values. However, while q-functions offer solid theoretical guarantees in tabular RL, they can be unstable in Deep RL. When paired with bootstrapping and off-policy learning, q-values are well known to diverge from the optimal solution (Sutton & Barto, 2018). van Hasselt et al. (2018) provide empirical evidence of the phenomenon, investigating the relationship between divergence and performance, and how different variables affect divergence. In particular, the work shows that the DQN (Mnih et al., 2015) is not guaranteed to converge to the optimal q-function. The divergence rate on both evaluation and control problems increases depending on specific mechanisms, such as the amount of bootstrapping, or the amount of prioritisation of updates (Schaul et al., 2015b).

An additional problem arises when employing GPI schemes to solve control problems. While during evaluation the policy is fixed, here the policy continuously changes. It becomes more challenging to track the target of the update while converging to it, as the change of policy makes the problem appear non-stationary from the point of view of the value estimation. In fact, even if the policy changes, there is no signal that informs the policy evaluation about the change. To mitigate the issue, many methods either use a fixed network as an evaluation target (Mnih et al., 2015), perform Polyak averaging of the target network (Haarnoja et al., 2018), or clip the gradient update to a maximum cap (Schulman et al., 2017). To further support the idea, theoretical and empirical evidence (Bellemare et al., 2016) shows that the q-function is *inconsistent*: for any suboptimal action a, the optimal value function $q^*(s, a)$ describes the value of a *non-stationary* policy, which selects a different action $\pi^*(s)$ (rather than a) at each visit of s.

The non-stationarity of q-values for suboptimal actions has also been shown empirically. Schaul et al. (2022) measure the per-state *policy change* $W(\pi, \pi'|s) = \sum_{a \in \mathcal{A}} |\pi(a|s) - \pi'(a|s)|$ for several Atari 2600 games Arcade Learning Environment (ALE) (Bellemare et al., 2013), and show that the action-gap undergoes brutal changes despite the agent maintaining a constant value of expected returns.

In practice, Deep RL algorithms often use q-targets to approximate the q-value, for example, n-step targets (Sutton & Barto, 2018, Chapter 7), or λ-returns (Watkins, 1989; Jaakkola et al., 1993; Sutton & Barto, 2018, Chapter 12). However, we consider them as *methods*, rather than quantities to measure credit, since they all ultimately aim to converge to the q-value. For this reason, we discuss them in Section 6.1.

Advantage (Baird, 1999) measures, in a given state, the difference between the q-value of an action and the value of its state

$$A^{\pi}(s,a) = q^{\pi}(s,a) - v^{\pi}(s). \tag{4.4}$$

Here, the context c is the same as in Equation (4.3). Because $v^{\pi}(s) = \sum_{a \in \mathcal{A}} q(s,a)\pi(a|s)$ and $A^{\pi}(s,a) = q^{\pi}(s,a) - \mathbb{E}_{\pi}[q^{\pi}(s,a)]$, the advantage quantifies how much an action is better than average.

As also shown in Bellemare et al. (2016), using the advantage to quantify credit can increase the *action-gap*. Empirical evidence has shown the consistent benefits of advantage over q-values (Baird, 1999; Wang et al., 2016b; Bellemare et al., 2016; Schulman et al., 2016), and the most likely hypothesis is its regularisation effects (Vieillard et al., 2020b;a; Ferret et al., 2021a). On the other hand, when estimated directly and not by composing state and state-action values, for example in Pan et al. (2022), the advantage does not permit bootstrapping. This is because advantage lacks an absolute measure of action influence, and only maintains one that is relative to the other possible actions.

Overall, in canonical benchmarks for both evaluation (Wang et al., 2016b) and control (Bellemare et al., 2013), advantage has been shown to improve over q-values (Wang et al., 2016b). In particular, policy evaluation experiences faster convergence in large action spaces because the state-value $v^{\pi}(s)$ can hold information that is shared between multiple actions. For control, it improves the score over several Atari 2600 games compared to both double q-learning (van Hasselt et al., 2016) and Prioritised Experience Replay (PER) (Schaul et al., 2015b).

General Value Functions (GVFs) (Sutton et al., 2011; Schaul et al., 2015a) are a set of q-value functions that predict returns for multiple reward functions:

$$q^{\pi,R}(s,a) = \{\mathbb{E}_{\mu,\pi}\left[\sum_t^T R(S_t,A_t)|S_t = s, A_t = a\right] : \forall R \in \mathcal{R}\}, \tag{4.5}$$

where R is a pseudo-reward function and $\mathcal{R}$ is an arbitrary, pre-defined set of reward functions. Notice that we omit the pseudo-termination and pseudo-discounting terms that appear in their original formulation (Sutton et al., 2011) to maintain the focus on credit assignment. The context c is the same of q-values and advantage, and the goal that the pseudo-reward represents is to reach a specific state $g = s \in \mathcal{S}$.

When first introduced (Sutton et al., 2011), the idea of GVFs stemmed from the observation that canonical value functions are limited to address only a single task at a time. Solving a new task would require learning a value function *ex-novo*. By maintaining multiple assignment functions at the same time, one for each goal, GVFs can instantly quantify the influence of an action for multiple goals simultaneously. However, while GVFs maintain multiple assignments, the goal is still not an explicit input of the value function. Instead, it is left implicit, and each assignment serves the ultimate goal to maximise a different pseudo-reward function (Sutton et al., 2011).

Universal Value Functions Approximators (UVFAs) (Schaul et al., 2015a) scale GVFs to Deep RL and advance their idea further by conflating these multiple assignment functions into a single one, represented as a deep neural network. Here, unlike for state-action values and GVFs, the goal is an explicit input of the assignment:

$$q^{\pi}(s,a,g) = \mathbb{E}_{\mu,\pi}[Z_t|S_t = s, A_t = a, G_t = g]. \tag{4.6}$$

The action influence here is measured for a goal explicitly. This allows to leverage the generalisation capacity of deep neural networks and to generalise not only over the space of states but also over that of goals.

Distributional values (Jaquette, 1973; Sobel, 1982; White, 1988; Bellemare et al., 2017) consider the full return distribution Z_t instead of its expected value:

$$Q^{\pi}(s, a) = \mathbb{P}_{\mu,\pi}(Z_t | S_t = s, A_t = a), \tag{4.7}$$

where $\mathbb{P}_{\mu,\pi}(Z_t)$ is the distribution over returns. Notice that we use uppercase Q to denote the value distribution and the lowercase q for its expectation (Equation (4.3)).

To translate the idea into a practical algorithm, Bellemare et al. (2017) proposes a discretised version of the value distribution by projecting $\mathbb{P}_{\mu,\pi}(Z_t)$ on a finite support $\mathcal{C} = \{0 \leq i \leq C\}$. The discretised value distribution then becomes $Q^{\pi}(s, a) = \mathbb{P}_{C}(Z_t | S_t = s, A_t = a)$, where $\mathbb{P}_C$ is a categorical Bernoulli that describes the probability that a return $c \in \mathcal{C}$ is achieved. Here, the context is the current MDP state and the goal is the expected return. Notice that while the optimal expected value function $q^*(s, a)$ is unique, in general, there are many optimal value distributions since different optimal policies can induce different value distributions.

Experimental evidence (Bellemare et al., 2017) suggests that distributional values provide a better quantification of the action influence, leading to superior results in well known benchmarks for control (Bellemare et al., 2013). However, it is yet not clear why distributional values improve over their expected counterparts. One hypothesis is that predicting for multiple goals works as an auxiliary task (Jaderberg et al., 2017), which often leads to better performance. Another hypothesis is that the distributional Bellman optimality operator proposed in Bellemare et al. (2017) produces a smoother optimisation problem, but the evidence remains weak or inconclusive (Sun et al., 2022).

Distributional advantage (Arumugam et al., 2021) proposes a distributional equivalent of the advantage:

$$A^{\pi}(s,a) = D_{KL}(Q^{\pi}(s,a)||V^{\pi}(s)), \tag{4.8}$$

and borrows the properties of both distributional values and the expected advantage. Intuitively, Equation (4.8) shows how much knowing the action changes the value distribution. To do so, it measures the change of the value distribution, for a given state-action pair, relative to the distribution for the particular state only. The KL divergence between the two distributions can then be interpreted as the distributional analogue of Equation (4.4), where the two quantities appear in their expectation instead. The biggest drawback of this measure of action influence is that it is only treated in theory, and there is no empirical evidence that supports distributional advantage as a useful proxy for credit in practice. Future works should consider providing empirical evidence on how this measure of action influence behaves compared to q-values and distributional values.

Hindsight advantage (Harutyunyan et al., 2019) stems from conditioning the action influence on future states or returns. The return-conditional hindsight advantage function can be written as follows:

$$A^{\pi}(s,a,z) = \left(1 - \frac{\mathbb{P}_{\pi}(A_t = a|S_t = s)}{\mathbb{P}_{\mu,\pi}(A_t = a|S_t = s, Z_t = z)}\right) z. \tag{4.9}$$

Here $A^{\pi}(s,a,z)$ denotes the return-conditional advantage and $\mathbb{P}_{\mu,\pi}(a_t|S_t = s, Z_t = z)$ is the return-conditional *hindsight distribution* and describes the probability that an action a has been taken in s, given that we observed the return z at the end of the episode, after following π. The context is a state, and the goal is the expected return, which, in this case, corresponds also to the value of the return collected in the current trajectory.

The idea of *hindsight* – initially presented in Andrychowicz et al. (2017) – is that even if the trajectory does not provide useful information for the main goal, it can be revisited as if the goal was the outcome just achieved. Hindsight advantage brings this idea to the extreme and rather than evaluating only for a pre-defined set of goals such as in Andrychowicz et al. (2017), it evaluates for every experienced state or return. Here, the action influence is quantified by that proportion of return determined by the ratio in Equation (4.9). To develop an intuition of it, if the action a is relevant to achieving the return z then $\mathbb{P}_{\mu,\pi}(A_t = a|S_t = s, Z_t = z) > \mathbb{P}_{\pi}(A_t|S_t = s)$, that is, the action is more likely to be taken in the hindsight distribution than in the behaviour policy, and the credit of the action is positive. Instead, if the action a is irrelevant to achieving the return z then $\mathbb{P}_{\mu,\pi}(A_t = a|S_t = s, Z_t = z) < \mathbb{P}_{\pi}(A_t|S_t = s)$, that is, the action is less likely to be taken in the hindsight distribution than in the behaviour policy, and the credit of the action is negative. When the action is irrelevant, then $\mathbb{P}_{\mu,\pi}(A_t = a|S_t = s, Z_t = z) = \mathbb{P}_{\pi}(A_t|S_t = s)$.

There exists also a state-conditional formulation rather than a return-conditional one, and we refer to Harutyunyan et al. (2019) for details on it to keep the description concise.

Future-conditional advantage (Mesnard et al., 2021) generalises hindsight advantage to use an arbitrary property of the future:

$$A^{\pi}(s, a, f) = \mathbb{P}_{\mu,\pi}(A_t = a|S_t = s, F_t = f)q^{\pi}(s, a, f). \tag{4.10}$$

Here, $F : \mathcal{D}^T \to \mathbb{R}^n$ is an n-dimensional feature of a trajectory d, and F_t is that feature for a trajectory that starts at time t and ends at the random horizon T. $q^{\pi}(s, a, f) = \mathbb{E}_{\mu,\pi}[Z_t|S_t = s, F_t = f, A_t = a]$ denotes the future-conditioned state-action value function. The context is a tuple of state and feature (s, f); the goal is the expected return observed at the end of the trajectory. Notice that you can derive the hindsight advantage by setting $F = Z$.

To develop an intuition, F can represent, for example, whether a day is rainy, and the future-conditional advantage expresses the probability of an action a, given that the day will be rainy.

Counterfactual advantage (Mesnard et al., 2021) proposes a specific choice of F such that F is independent of the current action. This produces a future-conditional advantage that factorises the influence of an action in two components: the contribution deriving from the intervention itself (the action) and the luck represented by all the components not under the control of the agent at the time t, such as fortuitous outcomes of the state-transition dynamics, exogenous reward noise, or future actions. The form is the same as that in Equation 4.10, with the additional condition that the feature F_t is independent of the action A_t and we have $\mathbb{E}_F[D_{KL}(\mathbb{P}(A_t|S_t = s)||\mathbb{P}(A_t|S_t = s, F_t = f))] = 0$.

The main intuition behind *counterfactual advantage* is the following. While to compute counterfactuals we need access to a model of the environment, in model-free settings we can still compute all the relevant information F_t that does not depend on this model. Once learned, a model of F can then represent a valid baseline to compute counterfactuals in a model-free way. To stay in the scope of this section, we detail how to learn this quantity in Section 6.4.

Posterior value functions (Nota et al., 2021) reflect on partial-observability and propose a characterisation of the hindsight advantage bespoke to POMDPs. The intuition behind Posterior Value Functions (PVFs) is that the evaluated action only accounts for a small portion of the variance of returns. The majority of the variance is often due to the part of the trajectory that still has to happen. For this reason, incorporating in the baseline information of the future could have a greater impact in reducing the variance of the policy gradient estimator. PVFs focus on the variance of a future-conditional baseline (Mesnard et al., 2021) caused by the partial observability. Nota et al. (2021) factorises a state s into an observable component o and an non-observable one u, and formalises the PVF as follows:

$$v_t^\pi(d) = \sum_{u \in \mathcal{U}} \mathbb{P}_{\mu,\pi}(U_t = u|d) v^\pi(o_t, u_t), \tag{4.11}$$

where $d = \{o_t, a_t, r_t : 0 \leq t \leq T\}$ is a trajectory comprising only observable components of the POMDP; $U_t \in \mathcal{U}$ is the non-observable component of a state S_t such that $s_t = \{u_t, o_t\}$.

Policy-conditioned values (Harb et al., 2020; Faccio et al., 2021) are value functions that include the policy as an input. For example, a policy-conditioned state-action value has the form:

$$q(s, \pi, a) = q^\pi(s, a), \tag{4.12}$$

but a representation of the policy π is used as an explicit input of the influence function. Here, the context is the union of the current MDP state s and the policy π, and the goal is the expected return at termination.

The main difference with state-action values is that, all else being equal, $q(s, \pi, a, g)$ produces different values *instantly* when π varies, since π is now an explicit input. For this reason, $q(s, \pi, a)$ can generalise over the space of policies, while $q^\pi(s, a)$ cannot. Using the policy as an input raises the problem of *representing* a policy in a way that can be fed to a neural network. Harb et al. (2020) and Faccio et al. (2021) propose two methods to represent a policy. To keep our attention on the CAP, we refer to their works for further details on possible ways to represent a policy (Harb et al., 2020; Faccio et al., 2021). Here we limit to convey that the problem of representing a policy has been already raised in the literature.

Name	Explicitness	Recursivity	Future-dependent	Causality
State-action value	○	●	○	○
Advantage	○	◒	○	○
GVFs/UVFAs	●	●	○	○
Distributional action-value	◒	●	○	○
Distributional advantage	◒	○	○	●
Hindsight advantage	◒	○	◒	○
Counterfactual advantage	◒	○	◒	●
Posterior value	○	○	●	○
Observation-action value	○	○	○	○
Policy-conditioned value	○	●	●	○

Table 4.2: A list of the most common *action influences* and their assignment functions in the Deep RL literature analysed in this survey, and the properties they respect. Respectively, empty circles, half circles and bullets indicate that the property is not respected, that it is only partially respected, and it is fully respected. See Sections 4.6 and 4.7 for details.

4.7 Discussion

The sheer variety of assignment functions described above leads to an equally broad range of metrics to quantify action influence. What is the best assignment function for a specific problem remains an open question. While we do not provide a definitive answer to the question of which properties are necessary or sufficient for an assignment function to output a satisfactory measure of credit, we set out to draw attention to the problem by abstracting out some of the properties that the metrics above share or lack. We identify the following properties of an assignment function and summarise our analysis in Table 4.2.

Explicitness. We use the term *explicitness* when the goal appears as an explicit input of the assignment and it is not left implicit or inferred from experience. Using the goal as an input allows generalising CA over the space of goals. The decision problem can then more easily be broken down into subroutines that are both independent of each other and independently useful to achieve some superior goal g.

Overall, explicitness allows incorporating more knowledge because the assignment spans each goal without losing information about others, only limited by the capacity of the function approximator. For example, UVFAs, hindsight advantages, and future conditional advantages are explicit assignments. As discussed in the previous section, *distributional values* can also be interpreted as explicitly assigning credit for each atom of the quantised return distribution, which is why we only partially consider them having this property in Table 4.2. Likewise, hindsight and future-conditional advantage, while not conditioning on a goal explicitly, can be interpreted as conditioning the influence on subgoals that are states or returns, and future statistics, respectively. For this reason, we consider them as partially explicit assignments.

Recursivity. We use the term *recursivity* to characterise the ability of an assignment function to support *bootstrapping* (Sutton & Barto, 2018). When an assignment is recursive, it respects a relationship of the type: $K(c_{t+1}, a_{t+1}, g) = f(K(c_t, a_t, g))$, where f projects the influence from the time t to $t + 1$. For example, goal-conditioned q-values can be written as: $q^\pi(s_{t+1}, a_{t+1}, g) = R(s_t, a_t, g) + \gamma q^\pi(s_t, a_t, g)$, where $R(s_t, a_t, g)$ is the reward function for the goal g.

Recursivity provides key advantages when *learning* credit, which we discuss more in detail in Section 6. In theory, it reduces the variance of the estimation at the cost of a bias (Sutton & Barto, 2018): since the agent does not complete the trajectory, the return it observes is imprecise but varies less. In practice, bootstrapping is often necessary in Deep RL when the length of the episode for certain environments makes full Monte-Carlo estimations intractable due to computational and memory constraints.

When the influence function does not support bootstrapping, the agent must obtain complete episodes to have unbiased samples of the return. For example, Direct Advantage Estimation (DAE) (Pan et al., 2022) uses the advantage function as a measure of credit, but it does not decompose the advantage into its recursive components that support bootstrapping ($q(s, a)$ and $v(s)$), and requires full Monte-Carlo returns to approximate it. This is often ill-advised as it increases the variance of the estimate of the return. For this reason, we consider the advantage to only partially satisfy recursivity.

Future-dependent. We use the term *future-dependent* for assignments that take as input information about what actions will be or have been taken *after* the time t at which the action A_t is evaluated. This is key because the influence of the current action depends also on what happens *after* the action. For example, picking up a key is not meaningful if the policy does not lead to opening the door afterwards.

Future actions can be specified *in-potentia*, for example, by specifying a policy to follow after the action. This is the case of policy-conditioned value function, whose benefit is to explicitly condition on the policy such that, if the policy changes, but the action remains the same, the influence of the action changes *instantly*. They can also be specified *in realisation*. This is the case, for example, of hindsight evaluations (Andrychowicz et al., 2017) such as the hindsight advantage, the counterfactual advantage, and the PVF where the influence is conditioned on some features of the future trajectory.

However, these functions only consider *features* of the future: the hindsight advantage considers only the final state or the final return of a trajectory; the counterfactual advantage considers some action-independent features of the future; the posterior value function considers only the non-observable components. Because futures are not considered fully, we consider these functions as only partially specifying the future.

Furthermore, while state-action value functions, the advantage and their distributional counterparts specify a policy in principle, that information is not an explicit input of the assignment, but only left implicit. In practice, in Deep RL, if the policy changes, the output of these assignments does not change unless retraining.

Causality. We refer to a *causal* assignment when the influence that it produces is also a measure of causal influence (Janzing et al., 2013). For example, the counterfactual advantage proposes an interpretation of the action influence closer to causality, by factorising the influence of an action in two. The first factor includes only the non-controllable components of the trajectory (e.g., exogenous reward noise, stochasticity of the state-transition dynamics, stochasticity in the observation kernel), or those not under direct control of the agent at time t, such as future actions. The second factor includes only the effects of the action alone. The interpretation is that, while the latter is due to causation, the former is only due to fortuitous correlations. This vicinity to causality theory exists despite the counterfactual advantage not being a satisfactory measure of causal influence as described in Janzing et al. (2013). Distributional advantage in Equation 4.8 can also be interpreted as containing elements of causality. In fact, we have that the expectation of the advantage over states and actions is the Conditional Mutual Information (CMI) between the policy and the return, conditioned on the state-transition dynamics: $\mathbb{E}_{\mu,\pi}[D_{KL}(Q^{\pi}(s,a)||V^{\pi}(s))] = \mathcal{I}(\mathbb{P}_{\pi}(A|S=s); Z|\mathbb{P}_{\mu}(S))$. The CMI (with its limitations (Janzing et al., 2013)) is a known measure of causal influence.

Overall, these properties define some characteristics of an assignment, each one bringing positive and negative aspects. Explicitness allows maintaining the influence of an action for multiple goals at the same time, promoting the reuse of information and a compositional onset of behaviour. Recursivity ensures that the influence can be learned via bootstrapping. Future-dependency separates assignments by whether they include information about future actions. Finally, causality filters out the spurious correlations evaluating the effects of the action alone.

4.7.1 Optimal credit assignment

Despite the literature invoking the notion of *optimal credit assignment* (e.g., (Raposo et al., 2021; Mesnard et al., 2023)), it seldom provides a formal definition. Formally unifying or defining these perspectives in precise mathematical terms lies beyond the scope of this thesis and would require extensive theoretical work. Here, we limit to providing a useful discussion on the topic, in the light of the framework we have just developed.

We identify two interpretations of the term "optimal credit assignment":

1. Optimality with respect to the *influence* function: the optimal influence function.
2. Optimality with respect to the optimal *assignment* function: the optimal assignment.

In the next paragraphs we briefly discuss the two interpretations, and we provide an intuition of what they could mean in the context of the CAP.

What is the "best" influence function is perhaps the underliying question behind studies proposing more effective *influence* functions. For example, the advantage function might be considered superior to q-values because it reduces the action-gap. The distributional advantage might be considered superior to the advantage because it reduces the variance of the estimate of the return. The hindsight advantage might be considered superior to the distributional advantage because it conditions the influence on the future. While we can determine a partial ordering over these measures of credit by measuring their impacts on the learning process (see Section 7), it remains hard to define a formal notion of optimality. Is the optimal influence function unique? Has it been already discovered.

Here we do answer to this question, but provide an intuition that derives from the framework described in this chapter. On one hand, a measure of credit is as good as it is effective to inform policy improvements. This means that the optimal influence function is the one that provides the most sample efficient, variance-free, and accurate improvements: the best estimate of the policy gradients. On the other hand, the best possible measure of a secondary reward signal to inform the agent about the quality of its actions is often discussed as the *optimal reward problem* (Sorg, 2011).

The second possible interpretation of the term "optimal credit assignment" is that of the best possible method to learn credit. That is, given an influence function, the optimal assignment is the one that converges faster, with less variance, and to the accurate influence. In pratice, an optimal assignment function is one that is able to learn the influence function with the least amount of data, in the shortest amount of time, and with the least amount of computational resources. The methods presented in Section 6 are all attempts to solve the CAP and to find an optimal assignment function.

4.8 Summary

In this section, we addressed **Question 1.1.** and discussed the problem of how to quantify action influences. In Section 4.1 we formalised our question: "*How do different works quantify action influences?*" and "*Are these quantities satisfactory measures of credit?*". We proceeded to answer the questions. In Section 4.2, we formalised the concept of *outcome* as some arbitrary function of a given history. In Section 4.3, we defined the assignment function as a function that returns a measure of action influence. In Section 4.4, we used this definition to formalise the CAP as the problem of learning a measure of action influence from experience. We refer to the set of protocols of this learning process as a credit assignment *method.* In Section 4.6, we surveyed existing measures of action influence from literature, detailed the intuition behind them, their advantages and drawbacks. Finally, in Section 4.7, we discussed how these measures of action influence relate to each other, the properties that they share and those that are rarer in literature, but still promising for future advancements. In the next sections, we proceed to address **Question 1.2.**. Section 5 describes the obstacles to solving the CAP and Section 6 surveys the methods to solve the CAP.

Chapter 5

The challenges to assign credit in Deep RL

Having clarified what measures of action influence are available in the literature, we now look at the obstacles that arise to learn them and, together with Section 6, answer **Question 1.2.**. We first survey the literature to identify *known issues* to assign credit and then systematise the relevant issues into CA challenges. These challenges provide a perspective to understand the principal directions of development of CA methods and are largely independent of the choice of action influence. However, using a measure of influence over another can still impact the prominence of each challenge.

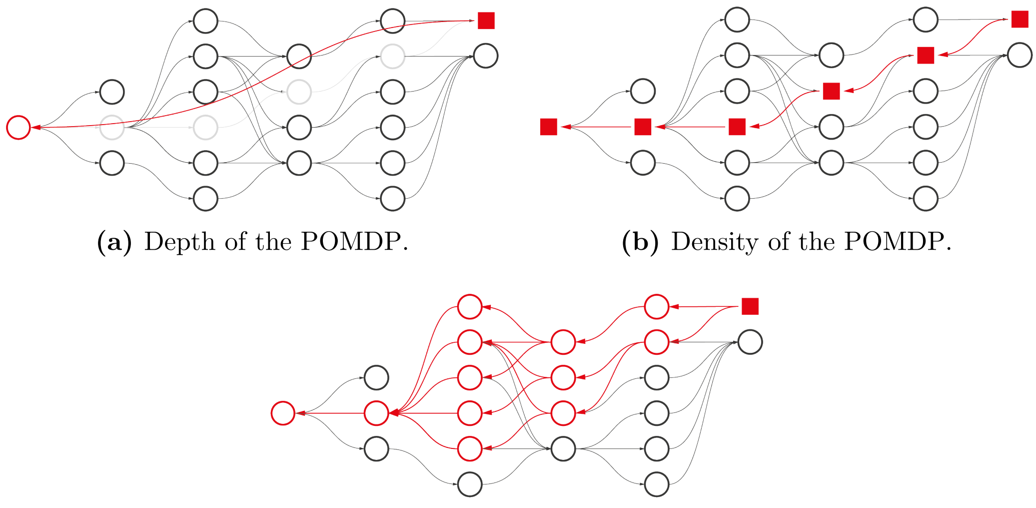
We identify the following issues to assign credit: *(a)* **delayed rewards** (Raposo et al., 2021; Hung et al., 2019; Arjona-Medina et al., 2019; Chelu et al., 2022): reward collection happens long after the action that determined it, causing its influence to be perceived as faint; *(b)* **sparse rewards** (Arjona-Medina et al., 2019; Seo et al., 2019; Chen & Lin, 2020; Chelu et al., 2022): the reward function is zero everywhere, and rarely spikes, causing uninformative TD errors; *(c)* **partial observability** (Harutyunyan et al., 2019): where the agent does not hold perfect information about the current state; *(d)* **high variance** (Harutyunyan et al., 2019; Mesnard et al., 2021; van Hasselt et al.,

2021) of the optimisation process; *(e)* the resort to **time as a heuristic** to determine the credit of an action (Harutyunyan et al., 2019; Raposo et al., 2021): *(f)* the lack of **counterfactual** CA (Harutyunyan et al., 2019; Foerster et al., 2018; Mesnard et al., 2021; Buesing et al., 2019; van Hasselt et al., 2021); *(g)* **slow convergence** (Arjona-Medina et al., 2019).

While these issues are all very relevant to the CAP, their classification is also tailored to control problems. Some of these are described by the use of a particular solution, such as *(e)*, or the lack thereof, like *(f)*, rather than by a characteristic of the decision or of the optimisation problem. Here, we systematise these issues and transfer them to the CAP. We identify three principal characteristics of POMDPs, which we refer to as *dimensions* of the POMDP: **depth**, **density** and **breadth** (see Figure 5.1). Challenges to CA emerge when pathological conditions on depth, density, and breadth mask the learning signal, making it unreliable, inaccurate, or insufficient to correctly reinforce an action. For example, very deep POMDPs can produce rewards that are delayed, while very sparse POMDPs can produce rewards that are improbable to encounter. We now detail these three dimensions and the corresponding challenges that arise.

5.1 Delayed effects due to high POMDP depth

We refer to the *depth* of a POMDP as the number of temporal steps that intervene between a highly influential action and an outcome (Ni et al., 2023). When this happens, we refer to the action as a *remote* action, and to the outcome as a *delayed* outcome. When outcomes are delayed, the increase of temporal distance often corresponds to a combinatorial increase of possible alternative futures and the paths to get to them. In these conditions, recognising which action was responsible for the outcome is harder, since the space



(a) Depth of the POMDP. **(b)** Density of the POMDP.

(c) Breadth of the POMDP.

Figure 5.1: Visual intuition of the three challenges to temporal CA and their respective set of solutions, using the graph analogy. Nodes and arrows represent, respectively, MDP states and actions. Blue nodes and arrows denote the current episode. Black ones show states that could have potentially been visited, but have not. Square nodes denote goals. Forward arrows (pointing right) represent environment interactions, whereas backward arrows (pointing left) denote credit propagation via state-action back-ups. From top left: **(a)** the temporal distance between the accountable action and the target state requires propagating credit deep back in time; **(b)** considering any state as a target increases the density of possible associations and reduces information sparsity; and finally, **(c)** the breadth of possible pathways leading to the target state.

of possible associations is very large. We identify two main reasons for an outcome to be delayed, depending on whether the decision after the remote action influences the outcome or not.

The first reason for delayed effects is that the success of the action is not immediate but requires a sequence of actions to be performed *afterwards*, which causes the causal chain leading to success to be long. This issue originates from the typical hierarchical structure of many POMDPs, where the agent must first perform a sequence of actions to reach a subjective sub-goal, and then perform another sequence to reach another. The key-to-door task (Hung et al., 2019) is a good example of this phenomenon, where the agent must first collect a key, to be able to open a door later.

The second reason is *delayed reinforcements*: outcomes are only *observed* after a long time horizon, and any decision taken *after* the remote action does not influence the outcome significantly. The phenomenon was first noted in behavioural psychology and is known as the *delayed reinforcement* problem (Lattal, 2010),

> *Reinforcement is delayed whenever there is a period of time between the response producing the reinforcer and its subsequent delivery.* (Lattal, 2010)

The main challenge with *delayed reinforcements* is in being able to ignore the series of irrelevant decisions that are encountered between the remote action and the delayed outcome, focus on the actions that are responsible for the outcome, and assign credit accordingly. This is a key requirement because most CA methods rely on temporal recency as a heuristic to assign credit (Klopf, 1972; Sutton, 1988; Mahmood et al., 2015; Sutton et al., 2016; Jiang et al., 2021a). When this is the case, the actions in the proximity of achieving the goal are reinforced, even if not actually being responsible for the outcome (only the remote action is), just because they are temporally close to the outcome.

While recent works advance proposals on how to measure the POMDP depth, for example, CA length (Ni et al., 2023), there is currently no formal agreement in the literature on how to diagnose the presence of delayed effects.

5.2 Low action influence due to low POMDP density

If delayed effects are characterised by a large temporal distance between an action and the outcome, POMDP sparsity derives from a *lack of influence* between them. Even if the literature often confounds *sparse* and *delayed* rewards, there is a substantial difference between them. With delayed effects, actions can cause outcomes very frequently, except with delay. Here, actions have little or no impact on the outcome, and outcomes do not vary regardless of the actions taken, but in a few, rare instances. We identify two main reasons.

The first one is highly stochastic state-transition dynamics, which can be diagnosed by measuring the entropy of the state-transition distribution $\mathcal{H}(\mathbb{P}_\mu)$ and/or of the reward function $\mathcal{H}(\mathbb{P}(R))$. In highly stochastic POMDPs, actions hardly affect the future states of the trajectory, the agent is unable to make predictions with high confidence, and therefore cannot select actions that are likely to lead to the goal.

The second reason is the low goal density. This is the canonical case of reward sparsity in RL, where the goal is only achievable in a small subset of the state space, or for a specific sequence of actions. Formally, we can measure the sparsity of an POMDP using the notion of information sparsity (Arumugam et al., 2021).

Definition 5 (POMDP sparsity)**.** *An POMDP is ε-information sparse if:*

$$\max_{\pi \in \Pi} \mathbb{E}_{\mu,\pi}[D_{KL}(P_{\mu,\pi}(Z|s,a)||P_{\mu,\pi}(Z|s))] \leq \varepsilon, \tag{5.1}$$

where $\mathbb{E}_{\mu,\pi}$ denotes the expectation over the stationary distribution induced by the policy and the state-transition dynamics. The information sparsity of a POMDP is the maximum information gain that can be obtained by an agent. When this is low everywhere, and only concentrated in a small subset of decisions, CA methods often struggle to assign credit, because the probability of behaving optimally is lower (Abel et al., 2021a), and there is rarely a signal to propagate.

5.3 Low action influence due to high POMDP breadth

We use the term *breadth* of an POMDP to denote the number of alternative histories h that produce the same outcome g. We then use the term *dilution* of credit, when many optimal pathways exist, and there is no *bottleneck* decision that the agent has to necessarily make to achieve the goal. We formalise the concept using the notion of the *null space* of a policy (Schaul et al., 2022):

$$\text{Null}(\pi) := \{\Omega | v^{\pi}(s) = v^{\pi'}(s)\} \quad \forall \pi, \pi' \in \Omega \subseteq \Pi, \forall s \in \mathcal{S}. \tag{5.2}$$

$\text{Null}(\pi)$ is the *null space* of a policy π, defined to be the subset of the space of all policies $\Omega \subset \Pi$ such that two policies $\pi, \pi' \in \Omega$ have the same expected state-value $v^{\pi}(s) = v^{\pi'}(s)$ in all the states of the POMDP $s \in \mathcal{S}$.

Credit dilution is often not a challenge for control because optimal behaviours are more probable. However, it can be problematic for CA. Most of the common baselines, such as Advantage Actor Critic (A2C) (Mnih et al., 2016) or PPO (Schulman et al., 2017), stop exploring after a small subset of optimal histories is found (or after a certain amount of time). Indeed, when $\text{diam}(\text{Null}(\pi^*))$ is large, there are many optimal histories. Yet, most of them are not included in the experience set $\mathcal{C}$ since exploration stopped prematurely, and credit will not be improved for those. This is particularly relevant for assignments that measure the influence of an action relative to another. For example, the advantage $A^{\pi}(s, a) = q^{\pi}(s, a) - \mathbb{E}_{a' \sim \pi}[q^{\pi}(s, a')]$ is inaccurate if $\mathbb{E}'_{a}[q^{\pi}(s, a)]$ is inaccurate, which requires taccurately evaluating $q, \forall a' \in \mathcal{A}$. This often results in a low diversity of behaviours (Parker-Holder et al., 2020), and a poor robustness to changes in the environment (Eysenbach & Levine, 2022).

5.4 Relationship with the POMDP's stochasticity

Up to this point, the three structural POMDP dimensions – *depth*, *density*, and *breadth* – were treated as if the environment were essentially deterministic. We now turn to the orthogonal issue of *stochasticity* and analyse how randomness in the transition and reward kernels perturbs the three formal objects introduced in Sections 4.3, 4.4: the **influence function** K, the **assignment operator** Λ, and the resulting **credit-assignment problem** (CAP).

Let $d \sim \mathbb{P}_\mu$ be a sampled transition, and $y_t = K^\theta(c_t, a_t, g_t)$ the target of the assignment operator Λ at time t, which, as defined in Section 4.3, is a random variable due to the stochasticity of the environment $\mathbb{P}_\mu$, and of the policy $\mathbb{P}_\mu$. Recall that we conflate the stochasticity of the reward function and the transition kernel into a single distribution $\mathbb{P}_\mu$. When either $P_\mu(s_{t+1}|s_t, a_t)$ or $P_\mu(r_{t+1}|s_t, a_t)$ has larges entropy $\mathcal{H}$, the variance of the random influence $\text{Var}[K]$ is high. The higher the variance, the more noisy are the estimate K^θ, and the more samples are required for K^θ to converge to the exact statistics of K.

Notice that conditions on depth, breadth and sparsity of the POMDP can contribute to the noise in the influence function. For example, in a sparse POMDP, the probability of observing a successful outcome is low, and the variance of the influence function is high. When $P_\mu(r_{t+1}|s_t, a_t)$ has high entropy, the highest possible information gain tends to decrease. This, in turns, decreases the information sparsity of the POMDP (Equation (5.1)), and exacerbates the problems discussed in Section 5.3. Likewise, in a deep POMDP, the increase in temporal distance between the action and the outcome increases the variance of the influence function, because the number of possible alternative histories increases. Finally, in a wide POMDP, where many alternative actions can lead to the same outcome, it is less probable that the agent will observe the same action leading to the same outcome, and the variance of the influence function increases.

5.5 Relationship with the exploration problem

One additional challenge in practical experiments is that it is often hard to disentangle the impacts of CA from those of exploration. In fact, discerning the effects of the two is often only done qualitatively. Here, we discuss the connection between the two problems, if they can be studied independently, and whether it is possible to find a way to diagnose and separate the effect of one from the other.

We use the interpretation of *exploration* as *the problem of acting in an unknown environment to discover temporal sequences of states, actions and rewards with the purpose of acquiring new information* (Amin et al., 2021; Jiang et al., 2023b). The acquired experiences then become part of the experience set $\mathcal{C}$, which is used to solve the CAP as described in Equation (4.2).

To visualise the difference between the exploration problem and the CAP, consider the usual key-to-door environment, where the agent needs to pick up a key, which opens a door, behind which lies a reward. While highly improbable (Abel et al., 2021a), this successful event is the result of chance and random behaviour[1]. Nevertheless, it is the responsibility of *exploration* to *discover* for the **first time** an optimal history, and to keep feeding the set $\mathcal{C}$ with useful discoveries. Then, once the successful experience $\mathcal{C}^*$ is in the set $\mathcal{C}$, it becomes the responsibility of the CA method to consume that experience and extract a measure of influence from the relationship context-action-outcome (Equation (4.1)) that supports effective improvements.

[1] Or, rather, by the laws dictated by the exploration algorithm.

This is a key difference because the very same behaviour has a different cause whether it comes from exploration or from CA. If due to exploration, it happens by chance, making it unlikely to occur again. If due to accurate CA, it is the result of informed decision-making, and funded on the ability to forecast (Sutton et al., 2011) the effects of an action. Then, when assignments start to be accurate enough, policy improvement further increases the probability of visiting optimal trajectories in a virtuous cycle that also improves CA. Many studies show how common RL baselines often struggle to extract a reliable signal from a small set of isolated successes. This is the case, for example, of A2C (Oh et al., 2018), DQN (Schaul et al., 2015b) or PPO (Arjona-Medina et al., 2019). To further support the claim, increasing the sampling probability of a success, for example through PER (Schaul et al., 2015b) or Self-Imitation Learning (SIL) (Oh et al., 2018), shows great improvements in CA.

We can draw two conclusions from the arguments above. On one hand, if there is a *minimum* number of optimal trajectories $\mathcal{C}^* \subset \mathcal{C}$ in $\mathcal{C}$, exploration has done its job and failures can be attributed to poor CA. On the other hand, a natural question arises: "*What is the minimum rate of successes $G_{min} = |\mathcal{C}^*|/|\mathcal{C}|$ that a CA method requires to start converging to an optimal policy?*". This is a fundamental open question in the current literature, and an answer to it can produce a valid tool to evaluate a CA method. All else being equal, the lowest the ratio $\mathcal{C}^*/\mathcal{C}$, the better the method, because it requires exploration to randomly collect optimal histories at a lower rate, and can solve harder POMDPs (Abel et al., 2021a).

5.6 Summary

In this section, we surveyed the literature and discussed both the obstacles and the current limitations to solving the CAP. These include delayed rewards, sparse rewards, partial observability, high variance, the lack of counterfactual CA, and sample efficiency. Then, we systematised these issues into challenges that emerge from specific properties of the decision problem, which we refer to as dimensions of the POMDP: depth, density, and breadth. Challenges emerge when pathological conditions on these dimensions produce specific phenomena that mask the learning signal to be unreliable, inaccurate, or insufficient to correctly reinforce an action: delayed effects, sparsity, and credit dilution. We have provided an intuition of this classification with the aid of graphs and proceeded to detail each challenge. Finally, we discussed the connection between the CAP and the exploration problem, suggesting a way to diagnose when a failure is caused by one or the other, and disentangling exploration from CA.

With these challenges in mind, we now proceed to review the state of the art in CA, and discuss the methods that have been proposed to address them.

Chapter 6

Learning to assign credit in Deep RL: existing methods

In this section, we propose a general form of a CA method such that we can survey existing algorithms, reconduce them to this form, and compare them.

What is the form a CA method? Following the definition of CAP in Section 4.4, a *credit assignment method* is then an algorithm that takes an initial guess $\widetilde{K}^{\phi} \in \mathcal{K}$ and a finite set of experience $\mathcal{D} = (\mathcal{S} \times \mathcal{A} \times \mathcal{R})^T$, and, by sampling and learning from transitions $D \sim \mathbb{P}_D$[1], it recursively produces a better approximation of the true assignment K.

We define a CA method according to how it specifies three elements:

(a) The influence function K.

(b) The assignment rule $\Lambda : \mathcal{K} \times \mathcal{D} \to \mathcal{K}$ that the method uses to approximate K^{θ}.

(c) The distribution $\mathbb{P}_D(d)$ sample from to collect experience data $d \in \mathcal{D}$.

[1] To enhance the flow of the manuscript, we formalise *contextual distributions* in Appendix A.2, and since they are intuitive concepts, we describe them in words when surveying the methods.

This provides consistency with the framework just proposed and allows categorising each method by the mechanisms that it uses to assign credit. Therefore, for each method, we report the three elements described above.

Taxonomy. Our classification aims to identify the principal directions of development and to minimise the intersection between each class of methods. We aim to understand the density around each set of approaches, to locate the branches suggesting the most promising results, and to draw a trend of the latest findings. This can be helpful to both the researchers on the CAP who want to have a bigger picture of the current state of the art, to general RL practitioners and research engineers to identify the most suitable methods to use in their applications, and to the part of the scientific community that focuses on different problems, but that can benefit from the insights on CA. We identify the following categories:

1. Methods using **time contiguity** as a heuristic (Section 6.1).
2. Those **decomposing returns** into per-timestep utilities (Section 6.2).
3. Those conditioning on **predefined goals** explicitly (Section 6.3).
4. Methods conditioning the present on **future outcomes in hindsight** (Section 6.4).
5. Modelling trajectories as **sequences** (Section 6.5).
6. Those **planning or learning backwards** from an outcome (Section 6.6).
7. **Meta-learning** different proxies for credit (Section 6.7).

We do not claim that this list is exhaustive. Rather, as in Section 4.6, this taxonomy is representative of the main approaches becoming a tool to understand the current state of the art in the field. We are keen to receive feedback on

Publication	Method	Class	Depth	Density	Breadth
Baird (1999)	AL	Time	○	○	●
Wang et al. (2016b)	DDQN	Time	○	○	●
Pan et al. (2022)	DAE	Time	○	○	●
Klopf (1972)	ET	Time	●	○	○
Sutton et al. (2016)	ETD	Time	●	○	○
Bacon et al. (2017)	Option-critic	Time	●	○	○
Hung et al. (2019)	TVT	Return decomposition	●	○	○
Arjona-Medina et al. (2019)	RUDDER	Return decomposition	●	○	○
Ferret et al. (2021a)	SECRET	Return decomposition	●	○	○
Ren et al. (2022)	RRD	Return decomposition	●	○	○
Raposo et al. (2021)	SR	Return decomposition	●	○	○
Sutton et al. (2011)	GVF	Auxiliary goals	○	●	○
Schaul et al. (2015a)	UVFA	Auxiliary goals	○	●	○
Andrychowicz et al. (2017)	HER	Future-conditioning	○	●	○
Rauber et al. (2019)	HPG	Future-conditioning	○	●	○
Harutyunyan et al. (2019)	HCA	Future-conditioning	○	●	○
Schmidhuber (2019)	UDRL	Future-conditioning	○	●	○
Mesnard et al. (2021)	CCA	Future-conditioning	○	●	●
Nota et al. (2021)	PPG	Future-conditioning	○	●	○
Janner et al. (2021)	TT	Sequence modelling	○	●	○
Chen et al. (2021)	DT	Sequence modelling	○	●	○
Goyal et al. (2019a)	Recall traces	Backward planning	○	●	●
Edwards et al. (2018)	FBRL	Backward planning	○	●	●
Nair et al. (2020)	TRASS	Backward planning	○	●	●
van Hasselt et al. (2021)	ET(λ)	Learning predecessors	●	○	●
Xu et al. (2018)	MG	Meta-Learning	●	○	○
Yin et al. (2023)	Distr. MG	Meta-Learning	●	○	○

Table 6.1: List of the most representative algorithms for CA classified by the CA challenge they aim to address. For each method, we report the publication that proposed it, the class we assigned to it, and whether it is designed to address each challenge described in Section 5. Hollow circles mean that the method does not address the challenge, and the full circle represents the opposite.

missing methods from the list to improve further revisions of the manuscript. We now proceed to describe the methods, which we also summarise in Table 6.1.

6.1 Time as a heuristic

One common way to assign credit is to use time contiguity as a proxy for causality: an action is as influential as it is temporally close to the outcome. This means that, regardless of the action being an actual cause of the outcome, if the action and the outcome appear temporally close in the same trajectory, the action is assigned high credit. At their foundation, there is TD learning (Sutton, 1988), which we describe below.

TD learning. (Sutton, 1984; 1988; Sutton & Barto, 2018) iteratively updates an initial guess of the value function according to the difference between expected and observed outcomes. More specifically, the agent starts with an initial guess of values, acts in the environment, observes returns, and aligns the current guess to the observed return. The difference between the expected return and the observed one is the TD error δ_t:

$$\delta_t = R(s_t, a_t) + \gamma \widetilde{q}^{\pi}(s_{t+1}, a_{t+1}) - \widetilde{q}^{\pi}(s_t, a_t) \tag{6.1}$$

where $a_{t+1} \sim \pi$ and $s_{t+1} \sim \mu$, and $\widetilde{q}^{\pi}$ denotes the current guess of the action value function. Because we focus on Deep RL, we omit the learning rate parameter. Notice that the reward $R(s_t, a_t)$ is the only part of the information grounded into the reality of the environment, while the rest of the information is the agent's guess. TD learning blends elements from Discounted Utility Theory (DUT) (Samuelson, 1937; Edwards, 1954; Simon, 1959a) where the credit is the expected future utility of an action, and from neuropsychology, where the error between the expected and the observed utility codes dopaminergic interactions (Schultz et al., 1997; Schultz, 1998; Montague et al., 1996). This strong temporal bias induces a propagation of the information that is *local* in time: the credit is propagated between actions that are adjacent in time and struggles to propagate to actions that are far apart in time.

Update rule. We can generalise the TD error to the difference of any influence function, following the notation in Section 4.6:

$$\delta_t^K = R(s_t, a_t) + \gamma K(s_{t+1}, a_{t+1}) - \widetilde{K}(s_t, a_t) \tag{6.2}$$

All methods in this section are characterised by the use of the this generalised

TD error to update their influence function, and their assignments follow this general structure of the update:

$$K(s_t, a_t) = \Lambda[K, d] = \widetilde{K}(s_t, a_t) + \delta_t^K = R(s_t, a_t) + \gamma K(s_{t+1}, a_{t+1}) \tag{6.3}$$

where $s_t, a_t, s_{t+1}, a_{t+1} \sim \mathbb{P}_D$.

When the temporal distance between the goal and the action is high – a premise at the base of the CAP – it is often improbable to observe very far rewards. As time grows, so does the variance of the observed outcome, due to the intrinsic stochasticity of the environment dynamics, and the policy. To mitigate the issue, TD methods often replace the theoretical measure of influence with an approximation: the *TD target*. In TD learning, the value function is updated to approximate the *target*, and not the theoretical measure of action influence underneath it. Since policy improvement uses the current approximation of the value to update the policy, future behaviours are shaped according to it, and the *TD target* drives the learning process.

We separate the methods in this category in three subgroups: those specifically designed around the advantage function, those re-weighing updates to stabilise learning, and those assigning credit to subsets of temporally extended courses of actions.

6.1.1 Advantage-based approaches

The first subset of methods uses the *advantage* (see Section 4.6) as a measure of action influence, but still uses time as a heuristic to learn it.

Actor-Critic (AC) methods with a baseline function (Sutton & Barto, 2018, Chapter 13) approximate the action influence using some estimator of the *advantage* function (Equation 4.4). In fact, the policy gradient is proportional to $\mathbb{E}_{\mu,\pi}[(Q^\pi(s,a) - b(s))\nabla \log \pi(a|s)]$ and if we choose $v^\pi(s)$ as our baseline

$b(s)$, we get $\mathbb{E}_{\mu,\pi}[(A^{\pi}(s,a))\nabla \log \pi(a|s)]$ because $q^{\pi}(s,a) - v^{\pi}(s,a) = A^{\pi}(s,a)$. The use of an action-independent baseline function usually helps to reduce the variance of the evaluation, and thus of the policy gradients, while maintaining an unbiased estimate of it (Sutton & Barto, 2018). What function to use as a baseline is the subject of major studies, and different choices of baselines often yield methods that go beyond using time as a heuristic (Harutyunyan et al., 2019; Mesnard et al., 2021; Nota et al., 2021; Mesnard et al., 2023).

Advantage Learning (AL) Baird (1999) also uses time as a proxy for causality. There are many instances of AL in the Deep RL literature. The Dueling Deep Q-Network (DDQN) (Wang et al., 2016b) improves on DQN by calculating the q-value as the sum between the state-value function and a normalised version of the advantage. Even if this results in using the q-value as a measure of action influence and $K(s,a) = v^{\pi}(s) + (A^{\pi}(s,a) - \sum_a A^{\pi}(s,a')/|\mathcal{A}|)$, approximating the advantage is a necessary step of it.

DAE (Pan et al., 2022) follows Wang et al. (2016b) with the same specification of the advantage but provides better connections between the advantage and causality theory. In particular, for fully observable POMDPs, the causal effect of an action a upon a scalar outcome G is defined as $\mathbb{E}[G|s,a] - \mathbb{E}[G|s]$. If we choose the return Z as outcome, this actually corresponds to the advantage $\mathbb{E}[Z|s,a] - \mathbb{E}[Z|s] = q^{\pi}(s,a) - v^{\pi}(s)$, which becomes an approximate expression for the causal influence of an action upon the random return, as discussed also in Arumugam et al. (2021). Here, the context is an POMDP state, the action is the greedy action with respect to the current advantage estimation, and the goal is the expected return at termination.

As explained in Section 5.3, advantage can be decomposed in two terms $A^{\pi}(s,a) = q^{\pi}(s,a) - v^{\pi}(s,a)$. Since $v^{\pi}(s) = \mathbb{E}_{\pi}[q^{\pi}(s,a)]$, it is clear that the accuracy of the advantage depends on the accuracy of the q-values of all actions. It has been shown that, because of this, estimating and incorporating the advantage in the q-value has a regularisation effect (Vieillard et al., 2020a).

Another effect is increasing the action-gap (i.e. the difference in value between the best and second-best action), which facilitates value learning. Because evaluations are more accurate for a greater portion of the state-action space, AL-based methods contribute to address POMDP breadth, as shown in Table 6.1.

6.1.2 Re-weighing updates and compound targets

The second subset of methods in this category re-weighs temporal updates according to some heuristics, which we detail below. Re-weighing updates can be useful to emphasise or de-emphasise important states or actions to stabilise learning in Deep RL (van Hasselt et al., 2018).

Eligibility Traces (ET) (Klopf, 1972; Singh & Sutton, 1996; Precup, 2000a; Geist et al., 2014; Mousavi et al., 2017) credit the long-term impact of actions on future rewards by keeping track of the influence of past actions on the agent's future reward. Specifically, an eligibility trace (Sutton & Barto, 2018, Chapter 12) is a function that assigns a weight to each state-action pair, based on the recency of the last visit to it. A *trace* $e_t(s)$ spikes every time a state (or state-action) is visited and decays exponentially over time until the next visit or until it extinguishes. At each update, the TD error, which determines the magnitude of the update, is scaled by the value of the trace at that state, and $\delta_t^{ET} = \delta_t e_t(s)$. There are several types of eligibility traces, depending on the law of decay of the trace. For example, with accumulating traces (Klopf, 1972), every visit causes an increment of the trace. Replacing traces (Singh & Sutton, 1996) are capped to a specific value, instead.

Deep Q(λ)-Networks (DQ(λ)Ns) (Mousavi et al., 2017) implement eligibility traces on top of DQN (Mnih et al., 2015). Here, the eligibility trace is a vector $e \in \mathbb{R}^d$ with the same number of components d as the parameters of the DNN, and the action influence is measured by the q-value with parameters set $\theta \in \mathbb{R}^d$. The context is an MDP state, the action is an off-policy action in a transition arbitrarily chosen from the buffer; the goal is the expected return. The ET information is embedded in the parameters θ since they are updated according to $\theta \leftarrow \theta + \delta e$. Here e is the eligibility trace, incremented at each update by the value gradient (Sutton & Barto, 2018, Chapter 12): $e \leftarrow \gamma\lambda e + \nabla_\theta q^\pi(s, a)$.

Finally, successive works advanced on the idea of ETs, and proposed different updates for the eligibility vector (Singh & Sutton, 1996; van Hasselt & Sutton, 2015; Precup, 2000a).

Emphatic Temporal Differences (ETDs) (Sutton et al., 2016; Mahmood et al., 2015; Jiang et al., 2021b) continue on the idea of ETs to weigh TD updates with a trace. They aim to address the issue that canonical ETs may suffer from early divergence when combined with non-linear function approximation and off-policy learning. The re-weighing in ETD is based on the *emphatic trace*, which encodes the degree of bootstrapping of a state.

Originating from tabular and linear RL, the intuition behind ETDs is that states with high uncertainty – the states encountered long after the state-action pair of evaluation – are more reliable, and vice versa. The main adaptation of the algorithm to Deep RL is by Jiang et al. (2021b), who propose the Windowed Emphatic TD(λ) (WETD) algorithm. In this approach, ETD is adapted to incorporate update windows of length n, introducing a mixed update scheme where each state in the window is updated with a variable bootstrapping length, all bootstrapping on the last state in the window. The influence of an action in WETD is the same as for any other ET, but the trace itself is different and measures the amount of bootstrapping of the current estimate.

ETDs provide an additional mechanism to re-weigh updates, the interest function $i : \mathcal{S} \rightarrow [0, \infty)$. By emphasising or de-emphasising the interest of a state, the interest function can be a helpful tool to encode the influence of the actions that had led to that state. Because hand-crafting an interest function requires human interventions, allowing suboptimal and biased results, Klissarov et al. (2022) proposes a method to learn and adapt the interest function at each update using meta-gradients. Improvements on both discrete control, such as ALE, and on continuous control problems, such as MuJoCo (Todorov et al., 2012), suggest that the interest function can be helpful to assign credit faster and more accurately.

Re-weighing updates includes a set of techniques to adjust the influence of past actions based on their temporal proximity to the current state. Such methods aim to mitigate the limitations of TD methods by dynamically adjusting the weight assigned to past actions, thereby emphasizing or de-emphasizing their contribution to future rewards. For this reason, these methods can be seen as potential solutions to mitigate the impacts of delayed effects and improve credit assignment in settings with high POMDP depth, as shown in Table 6.1.

6.1.3 Assigning credit to temporally extended actions

The third and last subset of methods in this category assigns credit to temporally extended actions rather than a single, atomic action. This is formalised in the *options framework* (Sutton et al., 1999; Precup, 2000b).

For the purpose of CA, *options*, also known as *skills* (Haarnoja et al., 2017; Eysenbach et al., 2018), can be described as the problem of achieving *sub-goals*, such that an optimal policy can be seen as the composition of elementary behaviours. For example, in a key-to-door environment, such as MiniGrid (Chevalier-Boisvert et al., 2018) or MiniHack (Samvelyan et al., 2021) the agent might select the option *pick up the key*, followed by *open the door*. Each of this macro-action requires a lower level policy to be executed. For example, *pick up the key* requires selecting the actions that lead to reach the key before grabbing it. In the option framework, credit is assigned for each specific subgoal (the macro-action), with the benefits already described for the *explicitness* property, from Section 4.7. The idea stems from the intuition that it is easier to assign credit to macro-actions since a sequence of options is usually shorter than a sequence of atomic actions, reducing the overall temporal distance to the time of achieving the goal.

However, since the option literature often does not explicitly condition on goals, but uses other devices to decompose the CA problem, we review works about learning options next, and dedicate a separate section to auxiliary goal-conditioning in Section 6.3.

The option-critic architecture (Bacon et al., 2017) scales options to Deep RL and mirrors the actor-critic architecture but considering options rather than actions. The option-critic architecture allows learning both how to execute a specific option, and which option to execute at each time simultaneously and online. The option executes using the *call-and-return* model. Starting from a state s, the agent picks an option ω according to its policy over options π_Ω. This option then determines the primitive action selection process through the intra-policy π_ω until the option termination function β signals to stop. Learning options, and assigning credit to its actions, is then possible using the *intra-option policy gradient* and the *termination gradient* theorems (Bacon et al., 2017), which define the gradient (thus the corresponding update) for all three elements of the learning process: the option $\omega \in \Omega$, their termination function $\beta(s)$ and the policy over options π_Ω. Here, the context is a state $s \in \mathcal{S}$, the actions to assign credit to are both the intra-option action $a \in \mathcal{A}$ and the option $\omega \in \Omega$, and the goal is to maximise the return.

On the same lines, Riemer et al. (2018) propose *hierarchical option-critics*, which allows learning options at multiple hierarchical levels of resolution – nested options – but still only on a fixed number of pre-selected options. Klissarov & Precup (2021) further improve on this method by updating all options with a single batch of experience.

In the context of the option-critic architecture, CA occurs at multiple levels of the hierarchy. At the lower, intra-option level, where individual actions are taken, credit assignment involves determining the contribution of each action to the achievement of sub-goals. This is essential for learning effective policies for executing primitive actions within each option. At the higher level of the hierarchy, credit assignment involves attributing credit to options for achieving higher-level goals and involves identifying the contribution of each option to achieving the overall task objective. The hierarchical structure of the option-critic architecture facilitates credit assignment by decomposing the learning problem into multiple levels of abstraction. For their ability to decompose a bigger task into smaller sub-problems, these methods naturally improve credit assignment when effects are delayed and in settings with high POMDP depth (see Table 6.1).

6.1.4 Summary and discussion

The methods we covered in this section use the temporal distance between the context-action pair and a reward to measure the action influence. The closer is the action, the higher is its influence and vice versa. While this could maybe be a reasonable assumption when the policy is optimal, it is not the case for the early exploratory stages of learning. In fact, as described in Section 5.5, highly influential actions are often taken long before their rewards are collected while exploring. For example, in our usual key-to-door example, the agent would pick up the key, perform hundreds of random, unnecessary actions, and the goal-tile only reached after those. In these cases, the two events are separated by a "*multitude of* [random and non-influential] *decisions*" (Minsky, 1961). Because these non-influential actions are temporally closer to reaching the goal-tile than that of picking up the key, these methods mistakenly assign them high influence and, in particular, a higher influence than to pick up the key.

Today, methods that assign credit only by looking at the temporal distance between the action and the outcome usually underperform on tasks with delayed effects (Arjona-Medina et al., 2019). Nevertheless, some of the branches in this category improve assignments in condition of high POMDP depth by re-weighing updates, using advantage or breaking down the task into multiple, composable subtasks.

6.2 Decomposing return contributions

To improve CA in settings with high POMDP depth, the line of research we describe next focuses on decomposing returns into per-timestep contributions. These works interpret the CAP as a *redistribution* problem: the return observed at termination is re-distributed to each time-step with an auxiliary mechanism that depends on each method and complements TD learning.

Temporal Value Transport (TVT) (Hung et al., 2019) uses an external long-term memory system to improve on delayed tasks. The memory mechanism is based on the Differentiable Neural Computer (DNC) (Grefenstette et al., 2015; Graves et al., 2016), a neural network then reads events from an external memory matrix, represented as the hidden state of a Long-Short-Term-Memory (LSTM). The agent decides to read from and write into it. To write, state-action-reward triples are projected to a lower dimensional space, and processed by the DNC. During training, this works as a trigger: when a past state-action pair is read from memory, it gets associated with the current one, transporting the state-action value – credit – from the present to the remote state. To read, the state-action-reward is reconstructed from the latent code. During inference, this acts as a proxy for credit. If a past state-action-reward triple is retrieved from the memory, it means that it is correlated with the current return. This allows to use the retrieval score of a past transition as a measure of the influence of its action.

Return Decomposition for Delayer Rewards (RUDDER) (Arjona-Medina et al., 2019) stems from the intuition that, if we can construct a reward function that *redistributes* the rewards collected in a trajectory such that the expected future reward is zero, we obtain an instantaneous signal that immediately informs the agent about future rewards. The method proposes to learn an influence function $f : (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$ such that, given the state-action pairs, it outputs the sum of discounted rewards, including the past, present and future rewards:

$$f(s_t, a_t) = \sum_{k=0}^{T} \gamma^k R(s_k, a_k) \tag{6.4}$$

Notice that this is different from an action-value function, as it also includes past rewards. In practice, f is implemented as an LSTM, which is trained to fit a subset of the whole experience set $\mathcal{D}_r \subset \mathcal{D}$. $\mathcal{D}_r$ is constructed to contain only trajectories containing delayed rewards, and experience is sampled proportionally to the current prediction error. The underlying hypothesis is that, by fitting the return, the LSTM's hidden state holds useful information to redistribute the return to the most relevant transitions in the sequence.

Once K is accurate enough, at each iteration of the RL algorithm, RUDDER employs contribution analysis to redistribute the sum of rewards to each state-action pair. It uses the difference between the inferred returns (i.e., the redistributed returns) at two consecutive time steps as a reward to perform canonical TD learning. Because two consecutive predictions share the same past and the same future contributions, except for two immediate state-action pairs, the difference of consecutive predictions contributions cancels out but for the two immediate state-action pairs.

The corresponding assignment is straightforward:

$$\widetilde{K}(s_t, a_t) = \Lambda[\widetilde{K}, d] = f(s_{t+1}, a_{t+1}) - f(s_t, a_t) \tag{6.5}$$

Self-Attentional Credit Assignment for Transfer (SECRET) (Ferret et al., 2021a) uses a causal Transformer-like architecture (Vaswani et al., 2017) with a self-attention mechanism (Lin et al., 2017) in the standalone supervised task of reconstructing the sequence of rewards from observations and actions. It then views attention weights over past state-action pairs as credit for the generated rewards. This was shown to help in settings of high POMDP depth in a way that transfers to novel tasks when trained over a distribution of tasks. We can write its measure of action influence as follows:

$$K(s_t, a_t) = \sum_{t=1}^{T} \mathbb{1}\{S_t = s, A_t = a\} \sum_{i=t}^{T} \alpha_{t \leftarrow i} R(s_i, a_i). \tag{6.6}$$

Here, $\alpha_{t \leftarrow i}$ is the attention weight on (o_i, a_i) when predicting the reward r_j.

Synthetic returns (SR) (Raposo et al., 2021) assume only one state-action to be responsible for the terminal reward. They propose a form of state pairs association where the earlier state (the *operant*) is a leading indicator of the reward obtained in the later one (the *reinforcer*). The association model is learned with a form of episodic memory. Each entry in the memory buffer, which holds the states visited in the current episode, is associated with a reward – the *synthetic* reward – via supervised learning. At training time, this allows propagating credit *directly* from the reinforcer to the operant, bypassing the local temporal difference. When this reward model is accurately learned, each time the operant is observed, the synthetic reward model spikes, indicating a creditable state-action pair. Here the synthetic reward acts as a measure of causal influence, and we write:

$$\widetilde{K}(s_t, a_t) = \Lambda[\widetilde{K}, d] = q^{\pi}(s_t, a_t) + f(s_t). \tag{6.7}$$

Here $f(s)$ is the synthetic reward function, and it is trained with value regression on the loss $||r_t - u(s_t) \sum_{k=0}^{t-1} f(s_t) - b(s_t)||^2$, where $h(s_t)$ and $b(s_t)$ are auxiliary neural networks optimised together with f. As for Arjona-Medina et al. (2019), the context c is a history h from the assigned POMDP, the action is an action from the trajectory $a \in h$, and the goal is the achieved return. This method is, however, stable only within a narrow range of hyperparameters and assumes that only one single action is to be credited.

6.2.1 Summary and discussion

The methods in this section assign credit by decomposing returns into per time-step contributions and then learning values from this new, clearer reward signal. For the purposes of this survey, they mainly differ by the method used to redistribute the contributions to each context-action pair. TVT uses an external memory system, RUDDER uses contribution analysis, SECRET exploits the Transformer's self-attention weights, SR use a gating function. Their motivation stems from improving on delayed effects, which they often state as an explicit goal, and for this reason, we report them as improving CA in settings of high POMDP depth (Table 6.1).

Indeed, the empirical evidence they provide suggests that improvements are consistent, and *redistribution* methods provide benefits over their TD learning baselines. On the other hand, these methods do not provide formal guarantees that the assignments improve over TD learning, and there is currently a gap to fill to justify these improvements also theoretically. This is the case, for example, of other methods (Harutyunyan et al., 2019; Wang et al., 2016b; Mesnard et al., 2021; van Hasselt et al., 2021) that prove to reduce the variance of the evaluation, some of which we describe in later sections.

6.3 Conditioning on a predefined set of auxiliary goals

The methods in this category evaluate actions for their ability to achieve multiple goals explicitly. They do so by conditioning the value function on a goal and then using the resulting value function to evaluate actions. The intuition behind them is that the agent's knowledge about the future can be decomposed into more elementary associations between states and goals.

Their assignment follow the same temporal structure of those in **Equation 6.3**, despite using different influence functions. We now describe the two most influential methods in this category.

General Value Functions (GVFs) (Sutton et al., 2011), described in Section 4.6, stem from the idea that knowledge about the world can be expressed in the form of predictions. These predictions can then be organised hierarchically to solve more complex problems. While GVFs carry several modifications to the canonical value, we focus on its goal-conditioning for the purpose of this review, which is also its foundational idea. GVFs conditions the action value on a goal to express the expected return with respect to the reward function that the goal induces. In their original formulation (Sutton et al., 2011), GVFs are a set of value functions, one for each goal. The goal is any object in a predefined goal set of MDP states $g \in \mathcal{S}$, and the resulting measure of action influence is the following:

$$K(s, a, g) = q^{\pi}(s, a, g), \tag{6.8}$$

that is the q-function with respect to the goal-conditioned reward function $R(s, a, g)$, which is 0 everywhere, and 1 when a certain state is reached. Because GVFs evaluate an action for what it is going to happen in the future, GVFs are forward methods, and interpret the CAP as a prediction problem: "What is the expected return of this action, given that g is the goal?".

Universal Value Functions Approximators (UVFAs) (Schaul et al., 2015a) scale the idea of GVFs to a large set of goals, by using a single value function to learn the whole space of goals. One major benefit of UVFAs over GVFs is that they are readily applicable to Deep RL by simply adding the goal as an input to the value function approximator. This allows the agent to learn end-to-end with bootstrapping and allows for exploiting a shared prediction structure across different states and goals. Since they derive from GVFs, UVFA share most of their characteristics. The context is an MDP state $s \in \mathcal{S}$; the goal is still any object in a predefined goal set of states, $g \in \mathcal{S}$, and the credit of an action is the expected return of the reward function induced by the goal (see Equation (6.8)).

6.3.1 Summary and discussion

The methods in this category stand out for using an *explicit* goal to assign credit, as described in Section 4.7. What distinguishes these methods from those that follow in the next section (which also use goals explicitly) is their flexibility. While in hindsight methods choose the goal after completing a trajectory, or based on information acquired during training, these methods do not. Instead, the set of goals of a GVF is predefined in Sutton et al. (2011). UVFAs, even if they can generalise to new goals in theory, they are designed with that purpose in mind, and their application is limited. This represents both a strong limitation of these methods, and a gap to fill in the literature, since it limits both their flexibility and their autonomy to adapt to different tasks, requiring the human designer to specify the set of goals *ex-ante* and to provide the set of goals as an input at the start of training.

Furthermore, their interpretation of credit is still linked to the idea of temporal contiguity described in Section 6.1. For this reason, they share many drawbacks and limitations with those methods and perform poorly when the POMDP is deep, especially if not accompanied by more advanced techniques. To the best of our knowledge, there are no examples in the literature that pair these methods with more advanced CA techniques (e.g., *options*), which represents a gap to fill.

On the other hand, by specifying a goal explicitly (GVFs) and by generalising over the goal space (UVFA), conditioning on a predefined set of goals provides a way to extract signals from the environment even when the signal is sparse and action influence is low. In fact, even when the main task is complex, the set of auxiliary goals is designed to provide a useful signal for learning. This is the reason why we consider these methods improving CA when the POMDP is sparse (see Table 6.1).

6.4 Conditioning in hindsight

The methods in this category are characterised by the idea of re-evaluating the action influence according to what the agent achieved, rather than what it was supposed to achieve. This means that, given a trajectory h, we can choose some goal $g \in \mathcal{G}$ (*after* collecting h) and evaluate the influence of all the actions in h upon achieving g.

We separate the methods in this category into three subgroups.

(i) Those that re-label the past experience under a different perspective, such as achieving a different goal than the one the agent started.

(ii) Those that condition the action evaluation on some properties of the future during training, which becomes an explicit performance request at inference time.

(iii) Those that condition on future factors that are independent on the evaluated action, but that still influence future returns.

6.4.1 Relabelling experience

Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) stems from the problem of learning in sparse rewards environments, which is an example of low action influence in our framework (see Section 5.2. The method exploits the fact that even if a trajectory is suboptimal for the overall implicit goal to maximise MDP returns, it can be viewed as optimal if the goal is to achieve its final state.

In practice, HER brings together UVFAs and experience replay (Lin, 1992) to re-examine trajectories. After collecting a set of trajectories from the environment, the agent stores each transition in a replay buffer, together with both the state it sought to reach and the one that it actually did reach. This allows optimising $\widetilde{K}^{\phi}(s, a, g)$ for both goals. We refer to this process of re-examining a trajectory collected with a prior goal in mind and evaluating it according to the actually realised outcome as *hindsight conditioning*, which is also the main innovation that HER brings to the CAP. Notice that the original goal is important because the trajectory is collected with a policy that aims to maximise the return for that specific goal.

However, in HER, the goal set is still predefined, which requires additional specifications from the agent-designer and can limit the autonomy of the overall agent, which increases the autonomy of the agent. HER uses the goal-conditioned q-values described in Section 6.3 to measure action influence:

$$K(s_t, a_t, s_T) = q^{\pi}(s_t, a_t, s_T). \tag{6.9}$$

Here the context is a history from the POMDP, the action is an action from the trajectory $a \in h$, and the goal g is to visit a state s_T at the end of a trajectory.

Since HER is limited to off-policy learning with experience replay, **Hindsight Policy Gradients (HPGs)** (Rauber et al., 2019) transfers the findings of HER to Policy Gradient (PG) methods, and extend it to online settings. Instead of updating the policy based on the actual reward received, Hindsight Policy Gradient (HPG) updates the policy based on the hindsight reward, which is calculated based on the new goals that were defined using HER. The main difference with HER is that in HPGs, both the critic and the actor are conditioned on the additional goal. This results in a goal-conditioned policy $\pi(\cdot|S = s, G = g)$, describing the probability of taking an action, given the current state and a realised outcome. The action influence used in HPG is the advantage formulation of the hindsight policy gradients:

$$K(s, a, g) = q^{\pi}(s, a, g) - v^{\pi}(s, g), \tag{6.10}$$

where $q^{\pi}(s, a, g)$ and $v^{\pi}(s, g)$ are the goal-conditioned value functions. Here the context c is a history $h = \{o_t, a_t, r_t : 0 \leq t \leq T\}$, the goal is arbitrarily sampled from a goal set, $g \in \mathcal{G}$. Like HER, HPG is tailored to tasks with low action influence due to low MDP density, and it is shown to be effective in sparse reward settings. Overall, HER and HPG are the first completed work to talk about *hindsight* as the re-examination of outcomes for CA. Their solution is not particularly interesting for the CAP as they do not cast their problem as a CAP and they do not connect the finding to the CAP explicitly. However, they are key precursors of the methods that we review next, which instead provide novel and reusable developments for CAP specifically.

6.4.2 Conditioning on the future

Hindsight Credit Assignment (HCA) Traditional reinforcement learning algorithms often struggle with credit assignment as they rely solely on foresight: they evaluate actions against a predetermined goal, selected *before* acting. These methods operate under the assumption that we lack knowledge of what occurs beyond a given time step, making accurate credit assignment challenging, especially in tasks with delayed effects. (Harutyunyan et al., 2019), on the other hand, centres on utilising hindsight information, acknowledging that credit assignment and learning typically take place after the agent completes its current trajectory. This approach enables us to leverage this additional data to refine the learning of critical variables necessary for credit assignment.

(Harutyunyan et al., 2019) introduces a new family of algorithms known as Hindsight Credit Assignment (HCA). Hindsight Credit Assignment (HCA) algorithms explicitly assign credit to past actions based on the likelihood of those actions having been taken, given that a certain outcome has been observed. This is achieved by comparing a learned *hindsight distribution* over actions, conditioned by a future state or return, with the policy that generated the trajectory.

More precisely, the *hindsight distribution*, $h(a|s_t, \pi, g)$ is the likelihood of an action a, given the outcome g experienced in the trajectory $d \sim \mathbb{P}_{\mu,\pi}(D|S_0 = s, a_t \sim \pi)$. In practice, Harutyunyan et al. (2019) consider two classes of outcomes: states and returns. We refer to the algorithms that derive from these two classes of goals as *state-HCA* and *return-HCA*. For state-HCA, the context c is the current state s_t at time t; the outcome is a future state in the trajectory $s_{t'} \in d$ where $t' > t$; the credit is the ratio between the state-conditional hindsight distribution and the policy $\frac{h_t(a|s_t, s'_t)}{\pi(a|s_t)}$. For return-HCA, the context c is identical; the outcome is the observed return Z_t; the credit is the ratio between the return-conditional hindsight distribution and the policy $1 - \frac{\pi(a|s_t)}{h_t(a|s_t, Z_t)}$. The resulting ratios provide a measure of how crucial a particular action was in achieving the outcome. A ratio deviating further from 1 indicates a greater impact (positive or negative) of that action on the outcome. For example, return-HCA measures the influence of an action with the *hindsight advantage* described in Section 4:

$$K(s_t, a_t, z_t) = \left(1 - \frac{\pi(a_t|S_t = s_t)}{\mathbb{P}_{\mu,\pi}(a_t|S_t = s_t, Z_t = z_t)}\right) z_t. \tag{6.11}$$

To compute the *hindsight distribution*, HCA algorithms employ a technique related to importance sampling. Importance sampling estimates the expected value of a function under one distribution (the *hindsight distribution*) using samples from another distribution (the policy distribution). In the context of HCA, importance sampling weights are determined based on the likelihood of the agent taking each action in the trajectory, given the hindsight state compared to the likelihood of the policy for that same action. Once the hindsight distribution is computed, HCA algorithms can be used to update the agent's policy and value function. One approach involves using the hindsight distribution to reweight the agent's experience. This means the agent will learn more from actions that were more likely to have contributed to the observed outcome.

Besides advancing the idea of hindsight, (Harutyunyan et al., 2019) carries one novelty: the possibility to drop the typical policy evaluation settings, where the goal is to learn a value function by the repeated application of the Bellman expectation backup. Instead, action values are defined as a measure of the likelihood that the action and the outcome appear together in the trajectory, and are a precursor of the sequence modelling techniques described in the next section (Section 6.5).

Upside-Down RL (UDRL) (Schmidhuber, 2019; Srivastava et al., 2019; Ashley et al., 2022; Štrupl et al., 2022) is another implementation of the idea to condition on the future. The intuition behind Upside-Down RL (UDRL) is that rather than conditioning returns on actions, which is the case of the methods in Section 6.1, we can invert the dependency and condition actions on returns instead. This allows using returns as an input and inferring the action distribution that would achieve that return. The action distribution is approximated using a neural network, the *behaviour policy*, that is trained via maximum likelihood estimation using trajectories collected online from the environment. In UDRL the context is a completed trajectory d; the outcome is a command that achieves the return Z_k in $H = T - k$ time-steps, which we denote as $g = (Z_k, H)$; the credit of an action a is its probability according to the behaviour function, $\pi(a|s, g)$. In addition to HCA, UDRL also conditions the return to be achieved in a specific timespan.

Posterior Policy Gradients (PPGs) (Nota et al., 2021) further the idea of hindsight to provide lower-variance, future-conditioned baselines for policy gradient methods. At the base of PPG there is a novel value estimator, the PVF. The intuition behind PVFs is that in POMDPs the state value is not a valid baseline because the true state is hidden from the agent, and the observation cannot provide as a sufficient statistic for the return. However, after a full episode, the agent has more information to calculate a better, *a posteriori* guess of the state value at earlier states in the trajectory. Nota et al. (2021) refers to the family of possible *a posteriori* estimations of the state value as the PVF. Formally, a PVF decomposes a state into its current observation o_t, and some hidden state that is not observable and typically unknown b_t. The value of a state can then be written as the expected observation-action value function over the possible non-observable components $u_T \in \mathcal{U} = \mathbb{R}^d$. The action influence of a PPG is quantified by the expression:

$$K(d) = \mathop{\mathbb{E}}_{u \in \mathcal{U}} \left[\mathbb{P}(u_t = u | d) v(o_t, u_t) \right]. \tag{6.12}$$

Here, the context is an observation, the action is the current action and the goal is the observed return. In practice, PVF advances HCA by learning which statistics of the trajectory $\psi(d)$ are useful to assign credit, rather than specifying it objectively as a state or a return.

6.4.3 Exposing irrelevant factors

Counterfactual Credit Assignment (CCA) For being data efficient, credit assignment methods need to disentangle the effects of a given action of the agent from the effects of external factors and subsequent actions. External factors in reinforcement learning are any factors that affect the state of the environment or the agent's reward but are outside the agent's control. This can include things like the actions of other agents in the environment, changes in the environment state due to natural processes or events. These factors can make credit assignment difficult because they can obscure the relationship between the agent's actions and its rewards.

Mesnard et al. (2021) proposes to get inspiration from the counterfactuals from causality theory to improve credit assignment in model-free reinforcement learning. The key idea is to condition value functions on future events, and learn to extract relevant information from a trajectory. Relevant information here corresponds to all information that is predictive of the return while being independent of the agent's action at time t. This allows the agent to separate the effect of its own actions, *the skills*, from the effect of external factors and subsequent actions, the *luck*, which will enable refined credit assignment and therefore faster and more stable learning. It shows that these algorithms have provably lower variance than vanilla policy gradient, and develops valid, practical variants that avoid the potential bias from conditioning on future information. One variant explicitly tries to remove information from the hindsight conditioning that depends on the current action while the second variant avoids the potential bias from conditioning on future information thanks to a technique related to important sampling. The empirical evidence in Mesnard et al. (2021) suggests that CCA offers great improvements in tasks with delayed effects.

6.4.4 Summary and discussion

The methods described in this section bring many independent novelties to CA. The most relevant for our scope is the idea of hindsight conditioning, which can be summarised as evaluating past actions using additional information about the future, usually not available at the time the action was taken. They differ from those in Section 6.3, as they do not act on a pre-defined objective set of goals, but these are chosen *in hindsight*.

One drawback of these methods is that they must be able to generalise to a large goal space to be effective, which is not a mild requirement because the ability to generalise often correlates with the size of the network. This can limit the applicability of the method, especially in cases of low computation and memory budgets.

One of the greatest benefits of these methods is to always have a signal to learn from because, by construction, there is always a goal that has been achieved in the current trajectory, for example, the final state, or the terminal return. This, in turn, produces a higher number of context-action-outcome associations, translates into additional training data that is often beneficial in supervised problems, and results in an overall denser signal. These improvements in POMDPs with low density, which we report in Table 6.1, are supported by both empirical evidence and theoretical guarantees to reduce the variance of the evaluations (Harutyunyan et al., 2019; Wang et al., 2016b; Mesnard et al., 2021; van Hasselt et al., 2021). Incorporating information about the future (for example, future returns or states), is most likely one major reason why these algorithms overperform the others. In fact, when this information is designed to express particular features, such as action-independence or independence to irrelevant factors, such as in Mesnard et al. (2021), the gap increases even further.

Finally, some of these methods (Mesnard et al., 2021) also incentivise the discovery of multiple pathways to the same goal, by identifying decisions that are irrelevant to the outcome, resulting in the fact that any of them can be taken without affecting the outcome. The only requirement is to employ an actor-critic algorithm, which we consider a mild assumption, since transitioning from actor-critic to value-based settings is usually trivially achievable.

6.5 Modelling trajectories as sequences

The methods in this category are based on the observation that RL can be seen as a sequence modelling problem. Their main idea is to transfer the successes of sequence modelling in Natural Language Processing (NLP) to improve RL.

On a high level, they all share the same assumption: a sequence in RL is a sequence of transitions (s, a, r), and they differ in either how to model the sequence, the problem they solve, or the specific method they transfer from NLP.

Trajectory Transformers (TTs) (Janner et al., 2021) implements a decoder-only (Radford et al., 2018; 2019) Transformer (Vaswani et al., 2017) to model the sequence of transitions. TTs learn from an observational stream of data, composed of expert demonstrations resulting in an offline RL training protocol. The main idea of TTs is to model the next token in the sequence, which is composed by the next state, the next action, and the resulting reward. This enables planning, which TTs exploit to plan via beam search. Notice that, for any of these paradigms, if the sequence model is autoregressive – the next prediction depends only on the past history, but since a full episode is available, the future-conditioned probabilities are still well-defined, and also TTs can condition on the future. In TTs the action influence is the product between the action probability according to the demonstration dataset and its q-value:

$$K(s_t, a_t, z_t) = \mathbb{P}_\theta(A_t = a_t | Z_t = z_t) q^\pi(s_t, a_t). \tag{6.13}$$

Here, the context c is an MDP state $c = s \in \mathcal{S}$, the action is arbitrarily selected, and the goal is the return distribution $\mathbb{P}(Z)$.

Decision Transformers (DTs) (Chen et al., 2021) proceed on the same lines as TTs but ground the problem in learning, rather than planning. DTs interpret a sequence as a list of (s_t, a_t, Z_t) triples, where Z_t is the return-to-go. They then use a Transformer to learn a model of the actor that takes the current state and the return as input and outputs a distribution over actions. In addition, they optionally learn a model of the critic as well, which takes the current state and each action in the distribution to output the value of each action. The sequences are sampled from expert or semi-expert demonstrations, and the model is trained to maximise the likelihood of the actions taken by the expert. From the perspective of CA, TTs and DTs are equivalent, and they share the same limitation in that they struggle to assign credit accurately to experience beyond that of the offline dataset. Furthermore, like HCA (Harutyunyan et al., 2019), DTs bring more than one novelty to RL. Besides modelling the likelihood of the next token, they also use returns as input to the model, resulting in a form of future conditioning. However, for CA and this section, we are only interested in their idea of sequence modelling and we will not discuss the other novelties. There exist further extensions to DT both to online settings (Zheng et al., 2022) and to model quantities beyond the return (Furuta et al., 2022). The former allows assigning credit by modelling transition sequences in online settings. The latter, instead, generalises sequence modelling to transitions with additional arbitrary information attached – the same way, Future-Conditional Policy Gradient (FC-PG) generalise HCA.

6.5.1 Summary and discussion

Sequence modelling in RL transfers the advances in sequence modelling for NLP to Deep RL setting. The main idea is to measure credit by estimating the probability of the next action (or the next token), conditioned on the context and the goal defined in hindsight, according to an offline dataset of expert trajectories (Chen et al., 2021; Janner et al., 2021).

While some works propose adaptation to online fine-tuning (Lee et al., 2022), these methods mostly learn from offline datasets and the idea to apply sequence modelling online is underexplored. This represents a strong limitation as it limits the generalisation ability of these methods. For example, DT often fail to generalise to returns outside the training distribution.

The distribution that measures this likelihood $\mathbb{P}(a|c, g)$ can be interpreted as the hindsight distribution (Harutyunyan et al., 2018) described in Section 6.4. Their development has a similar pattern to that of hindsight methods and progressively generalises to more complex settings, such as online learning (Zheng et al., 2022) and more general outcomes (Furuta et al., 2022). In practice, these two trends converge together to model the likelihood of action, states and rewards, which hindsight methods call the *hindsight distribution*. Yet, this set of methods would benefit from a better connection to the RL theory. This has been the case for hindsight methods, which leverage notions from causality and the policy gradient theorem (Sutton & Barto, 2018) to achieve better experimental results (Mesnard et al., 2021). For the same reasons explained for hindsight methods in Section 6.4, these methods improve CA when the POMDP has low density and the action influence is low (see Table 6.1).

Nevertheless, sequence modelling remains a promising direction for CA, especially for their ability to scale to large datasets (Reed et al., 2022). It is not clear how these methods position with respect to the CA challenges described in Section 5, for the lack of experimentation on tasks that explicitly stress the agent's ability to assign credit. However, in their vicinity to future-conditioned methods, they bear some of the same advantages and also share some limitations. In particular, for their ability to define outcomes in hindsight, regardless of an objective learning signal, they bode well in tasks with low action influence.

6.6 Planning and learning backwards

The methods in this category extend CA to potential predecessor decisions that have not been taken, but could have led to the same outcome (Chelu et al., 2020). The main intuition is that, in environments with low action influence, highly influential actions are rare, and when a goal is achieved the agent should use that event to extract as much information as possible to assign credit to relevant decisions.

We divide the section into two major sub-categories, depending on whether the agent identifies predecessor states by planning with an inverse model, or by learning relevant statistics without it.

6.6.1 Planning backwards

Recall traces (Goyal et al., 2019a) combine model-free updates from Section 6.1 with learning a backward model of the environment. A backward model $\mu^{-1}(s_{t-1}|S_t = s, A_{t-1} = a)$ describes the probability of a state S_{t-1} being the predecessor of another state s, given that the action a was taken. This backward action is sampled from a *backward policy*, $\pi_b(a_{t-1}|s_t)$, which predicts the previous action, and a *backward dynamics*.

By autoregressively sampling from the backward policy and dynamics, the agent can cross the MDP backwards, starting from a final state, s_T, up until a starting state, s_0 to produce a new trajectory, called *recall trace*. This allows the agent to collect experience that always leads to a certain state, s_T, but that does so from different starting points, discovering multiple pathways to the same goal.

Formally, the agent alternates between steps of GPI via model-free updates and steps of behaviour cloning on trajectories collected via the backward model. Trajectories are reversed to match the forward arrow of time before cloning. This is a key step towards solving the CAP as it allows propagating credit to decisions that have not been taken but could have led to the same outcome without interacting with the environment directly. Recall-traces measure the influence of an action by its q-value, but differ from any other method using the same action influence because the contextual data is produced via backward crossing. The goal is to maximise the expected returns.

The same paradigm has been presented in a concurrent work (Edwards et al.,

2018) as Forward-Backward RL (FBRL). The benefits of a backward model have also been further investigated in other studies. Wang et al. (2021b) investigate the problem in offline settings, and show that backward models enable better generalisation than forward ones. van Hasselt et al. (2019) provide empirical evidence suggesting that assigning credit from hypothetical transitions, that is, via planning, improves the overall efficiency in control problems. Chelu et al. (2020) and van Hasselt et al. (2019) further show that backward planning provides even greater benefits than forward planning when the state-transition dynamics are stochastic.

6.6.2 Learning predecessors

Expected Eligibility Trace (ET(λ)) (van Hasselt et al., 2021) provide a model-free alternative to backward planning that assigns credit to potential predecessors decisions of the outcome: decisions that have been taken in the past but have not in the last episode. The main idea is to weight the action value by its expected eligibility trace, that is, the instantaneous trace (see Section 6.1), but in expectation over the random trajectory, defined by the policy and the state-transition dynamics.

The Deep RL implementation of ET(λ) considers the expected trace upon the action value representation – usually the last layer of a neural network value approximator. Like for other ETs algorithms, ET(λ) measures action influence using the q-value of the decision and encodes the information of the trace in the parameters of the function approximator. In this case, the authors interpret the value network as a composition of a non-linear representation function $\phi(s)$ and a linear value function $v(s_t) = w^\top \phi(s)$. The expected trace $e(s) = E\phi(s)$ is then the result of applying a second linear operator E on the representation. $e(s)$ is then trained to minimise the expected ℓ_2 norm between the current estimation of $e(s)$ and the instantaneous trace.

6.6.3 Summary and discussion

The methods in this section assign credit by considering the effects of decisions that have not been taken, but could have led to the same outcome. The intuition behind them is that, in tasks where the action influence is low due to low POMDP density, creditable actions are rare findings. When this happens the agent can use that occurrence to extract as much information as possible from them.

One set of methods does so by learning inverse models of the state-transition dynamics and walking backwards from the outcome. Chelu et al. (2020); van Hasselt et al. (2019) further analyse the conditions in which backward planning is beneficial. Another set of methods exploits the idea of eligibility traces and keeps a measure of the marginal state-action probability to assign credit to actions that could have led to the same outcome. Overall, these methods are designed to thrive in tasks where the action influence is low. Also, for their ability to start from a high-value state, backward planning methods can find a higher number of optimal transpositions, and therefore provide a less biased estimate of the credit of a state-action pair.

6.7 Meta-learning proxies for credit

The methods in this category aim to meta-learn key hyperparameters of canonical TD methods. In fact, RL methods are often brittle to the choice of hyperparameters, for example, the number of steps to look-ahead in bootstrapping, what discount factor to use, or meta-parameters specific to the method at hand. How to select these meta-parameters is an accurate balance that depends on the task, the algorithm, and the objective of the agent.

For this reason, it is sometimes difficult to analyse them using the usual framework, and we present them differently, by describing their main idea, and the way they are implemented in Deep RL.

Meta Gradient (MG) RL (Xu et al., 2018) remarks how different CA measures of action influence impact the performance on control problems, and proposes to answer the question: "*Among the most common TD targets, which one results in the best performance?*". The method interprets the target as a parametric, differentiable function that can be used and modified by the agent to guide its behaviour to achieve the highest returns.

In particular, *Meta-Gradients* consider the λ-return (Sutton, 1988) target, for it can generalise the choice of many targets (Schulman et al., 2016). It then learns its meta-parameters: the bootstrapping parameter λ and the discount factor γ. The connection between MG and CA is that, different pairs of meta-parameters evaluate actions differently. For example, changing the discount factor can move the focus of the assignment from early to late actions with effects on policy improvements (Xu et al., 2018). In fact, adapting and learning the meta-parameters online effectively corresponds to meta-learning a measure of action influence, and profoundly affects credit.

Meta-learning credit assignment strategies has been further extended to distributional (Yin et al., 2023) and continual (Zheng et al., 2020) settings. Badia et al. (2020) investigated the effects of meta-learning the discount factor and the exploration rate to balance out short and long-term rewards.

6.7.1 Summary and discussion

Overall, these methods assign credit to actions by applying canonical TD learning algorithms with a meta-learnt measure of action influence. The goal can come in the form of an update target (Xu et al., 2018; Zheng et al., 2018; Xu et al., 2020), a full return distribution (Yin et al., 2023), or a reward function (Zheng et al., 2020). This allows agents to adapt their influence function online, especially improving in conditions of high POMDP depth.

Chapter 7

Evaluating credit

Like accurate evaluation is fundamental to RL agents to improve their policy, an accurate evaluation of a CA method is fundamental to CA research to monitor if and how a method is advancing the field. The aim of this section is to survey the state of the art of the metrics, the tasks, and the evaluation protocols to evaluate a CA method. We discuss the main components of the evaluation procedure, the performance metrics, the tasks, and the evaluation protocols.

7.1 Metrics

We categorise existing metrics to evaluate a CA method in two main classes:

(a) The metrics that are already used for control problems. These mostly aim to assess the agent's ability to make optimal decisions, but they do not explicitly measure the accuracy of the action influence.

(b) The metrics that target the quality of an assignment directly, which usually aggregate metrics throughout the RL training procedure.

We now proceed to describe the two classes of metrics.

7.1.1 Metrics borrowed from control

Bias, variance and contraction rate. The first, intuitive, obvious proxy to assess the quality of a credit assignment method is its theoretical performance in suitable *control* problems: the bias, variance, and contraction rate of the policy improvement operator described in Rowland et al. (2020). Notice that these metrics are not formally defined for all the methods, either because some variables cannot be accessed or because the operators they act on are not formally defined for the method in question. For the evaluation operator described in Equation (2.21), we can specify these quantities as follows.

$$\Gamma = \sup_{s \in \mathcal{S}} \frac{||\mathcal{T}V^{\pi}(s) - \mathcal{T}V'^{\pi}(s)||_{\infty}}{||V^{\pi}(s) - V'^{\pi}(s)||_{\infty}} \tag{7.1}$$

is the contraction rate and describes how fast the assignment converges to its fixed point, if it does so, and thus how efficient it is. Here $V^{\pi}(s)$ and $V'^{\pi}(s)$ are two estimates of the state-value, which highlights that these set of metrics are not suitable to evaluate methods using any measure of action influence.

If $\mathcal{T}$ is contractive, then $\Gamma < 1 \, \forall V^{\pi}$ and V'^{π}, and there exist a fixed-point bias of $\mathcal{T}$ given by:

$$\xi = ||V^{\pi}(s) - \hat{V}^{\pi}(s)||_2, \tag{7.2}$$

where $\hat{V}^{\pi}(s)$ is the true, unique fixed point of $\mathcal{T}$, whose existence is guaranteed by $\Gamma < 1$. For every evaluation operator $\mathcal{T}$, there is an update rule $\Lambda : \mathbb{R}^{|\mathcal{S}|} \times \mathcal{H} \to \mathbb{R}$ that takes as input the current estimation of the state-value function, and a trajectory and outputs the updated function. Λ has a variance:

$$\nu = \mathbb{E}_{\mu,\pi}[||\Lambda[V(s), D] - \mathcal{T}V(s)||_2^2]. \tag{7.3}$$

These three quantities are usually in a trade-off (Rowland et al., 2020). Indeed, many (if not all) studies on credit assignment (Hung et al., 2019; Mesnard et al., 2021; Ren et al., 2022; Raposo et al., 2021) report the empirical return and its variance. Because the contraction rate is often harder to calculate, an alternative metric is the time-to-performance, which evaluates the number of interactions necessary to reach a given performance. These mostly aim at showing improvement in sample efficiency and/or asymptotic performance. While useful, this is often not enough to assess the quality of credit assignment, as superior returns can be the result of better exploration, better optimisation, better representation learning, luck (as per the environment dynamics' stochasticity) or of a combination of such factors. Using empirical returns makes the evaluation method empirically viable for any measure of action influence described in Section 4, even if these metrics are not formally defined for them. Nonetheless, when the only difference between two RL algorithms lies in how credit is assigned, and this is not confounded by the aforementioned factors, it is generally safe to attribute improvements to superior credit, given that the improvements are statistically significant (Henderson et al., 2018; Agarwal et al., 2021).

Task completion rate. A related, but more precise, metric is the success rate. Given a budget of trials, the success rate measures the frequency of task completion, that is, the number of times the task was solved over the total number of episodes: $G = |\mathcal{C}^*|/|\mathcal{C}|$. Here, $\mathcal{C}^*$ is a set of optimal histories experienced by the agent, and $\mathcal{C}$ is the full set of histories used to train it. Considering success rates instead of bias, variance, and trade-off is useful as it alleviates another issue of these performance metrics: there is no distinction between easy-to-optimise rewards and hard-to-optimise rewards. This is evident in the key-to-door task with distractors (Hung et al., 2019), which we describe in detail later in Section 7.2. Due to the stochasticity from the apple phase (the distractors), it is generally impossible to distinguish performance on apple picking (easy-to-optimise rewards) and door opening (hard-to-optimise rewards that superior credit assignment methods usually obtain). Furthermore, the minimum success rate G_{min} could also be an effective metric to disentangle the effects of exploration from those of CA as discussed in Section 5.5, despite never being employed for that purpose. However, notice that this clarity in reporting credit comes at a cost. In fact, even if these kinds of metrics are more precise than performance metrics, they require expert knowledge of the task. They often suffer from the same confounders as bias, variance, and contraction rate.

Value error. As the value function is at the heart of many credit assignment methods, another proxy for the quality of the credit is the quality of value estimation, which can be estimated from the distribution of TD errors (Andrychowicz et al., 2017; Rauber et al., 2019; Arjona-Medina et al., 2019). We can then generalise the value error to one of *influence error*: $\mathbb{E}[||\widetilde{K}(s,a,g) - K(s,a,g)||_i]$, where $||\cdot||_i$ denotes the ith norm of a vector, $\widetilde{K}(s,a,g)$ is the current approximation of influence and $K(s,a,g)$ is the true influence. A drawback of the influence error (and the value error) is that it can be misleading. When an algorithm does not fully converge, for example, because of high POMDP sparsity (see Section *(b)*, it can happen that the value error is very low. This is because the current policy never visits a state with a return different from zero, and the value function collapses to always return zero. Nevertheless, this metric is a viable option to evaluate RL methods that use some form of action influence. It is not applicable, for example, to PG methods using Monte-Carlo returns to improve a parametric policy via gradient ascent (Sutton & Barto, 2018), or to sequence modelling methods (see Section 6.5 that only approximate the action probabilities of a predefined set of demonstrations.

7.1.2 Bespoke metrics for credit assignments

We now review metrics that measure the quality of individual credit assignments, that is, how well actions are mapped to corresponding outcomes, or how well outcomes are redistributed to past actions. Usually, these metrics are calculated in hindsight, after outcomes have been observed.

Using knowledge about the causal structure. Suppose we have expert knowledge about the causal structure of the task at hand, i.e. which actions cause which outcomes. This is often the case since as humans we often have an instinctive understanding of the tasks agents tackle. In such a case, given an observed outcome from an agent's trajectory, one can compare credit assignments, which approximate such cause and effect relationships, to the ground truth represented by our causal model of the task. We give several examples from the literature. In Delayed Catch, Raposo et al. (2021) assess whether credit is assigned to the actions that lead to catches or to the end-of-episode reward since they know that these actions are causing the experienced rewards. They do the same on the Atari game Skiing, which is a more complex task but that shares the fact that only a subset of the actions of the agent yield rewards. For example, in Skiing, going between ski poles is the only thing that grants rewards (with delay) at the end of an episode. Ferret et al. (2021a) adopt a similar approach and look at the influence attributed to actions responsible for trigger switches in the Triggers environment, which contribute alone to the end-of-episode reward. Similarly, Arjona-Medina et al. (2019) look at redistributions of RUDDER on several tasks, including the Atari 2600 game Bowling.

Counterfactual simulation. A natural approach, which is nonetheless seldom explored in the literature, is counterfactual simulation. On a high level, it consists in asking what would have happened if actions that are credited for particular outcomes had been replaced by another action. This is close to the notion of hindsight advantage.

Comparing to actual values of the estimated quantity. This only applies to methods whose credit assignments are mathematically grounded, in the sense that they are the empirical approximations of well-defined quantities. In general, one can leverage extra compute and the ability to reset a simulator to arbitrary states to obtain accurate estimations of the underlying quantity, and compare it to the actual, resource-constrained quantity estimated from experience.

7.2 Tasks

In what follows, we present environments that we think are most relevant to evaluate credit assignment methods and individual credit assignments. The most significant tasks are those that present all three challenges to assign credit: delayed rewards, transpositions, and sparsity of the influence. This often corresponds to experiments that have reward delay, high marginal entropy of the reward, and partial observability. To benchmark explicit credit assignment methods, we additionally need to be able to recover the ground truth influence of actions w.r.t. given outcomes, or we can use our knowledge of the environment and develop more subjective measures.

7.2.1 Diagnostic tasks

Diagnostic tasks are useful as sanity checks for RL agents and present the advantage of running rather quickly, compared to complex environments with visual input that may imply several millions of samples before agents manage to solve the task at hand. Notice that these tasks may not be representative of the performance of the method at scale, but provide a useful signal to diagnose the behaviour of the algorithm in the challenges described in Section 5. Sometimes, the same environment can represent both a diagnostic task and an experiment at scale, simply by changing the space of the observations or the action space.

We first present chain-like environments, that can be represented graphically by a chain (environments **a** to **c**), and then a grid-like environment (environment **d**), that has more natural grid representations for both the environment and the state.

a) Aliasing chain. The aliasing chain (introduced in Harutyunyan et al. (2019) as Delayed Effect) is an environment whose outcome depends only on the first action. A series of perceptually aliased and zero-reward states follow this first action, and an outcome is observed at the end of the chain ($+1$ or -1 depending on the binary first action).

b) Discounting chain. The discounting chain (Osband et al., 2020) is an environment in which a first action leads to a series of states with inconsequential decisions with a final reward that is either 1 or $1 + \epsilon$, and a variable length. It highlights issues with the discounting horizon.

c) Ambiguous bandit. The ambiguous bandit (Harutyunyan et al., 2019) is a variant of a two-armed bandit problem. The agent is given two actions: one that transitions to a state with a slightly more advantageous Gaussian distribution over rewards with probability $1 - \epsilon$, and another that does so with probability ϵ.

d) Triggers. Triggers (Ferret et al., 2021a) is a family of environments and corresponding discrete control tasks that are suited for the quantitative analysis of the credit assignment abilities of RL algorithms. Each environment is a bounded square-shaped 2D gridworld where the agent collects rewards that are conditioned on the previous activation of all the triggers of the map. Collecting all triggers turns the negative value of rewards into positive and this knowledge can be exploited to assess proper credit assignment: the actions of collecting triggers appear natural to be credited. The environments are procedurally generated: when requesting a new environment, a random layout is drawn according to the input specifications.

7.2.2 Tasks at scale

In the following, we present higher-dimension benchmarks for agents equipped with credit assignment capabilities.

Atari. The Arcade Learning Environment (Bellemare et al., 2013) (ALE) is an emulator in which RL agents compete to reach the highest scores on 56 classic Atari games. We list the ones we deem interesting for temporal credit assignment assessment due to delayed rewards, which were first highlighted by Arjona-Medina et al. (2019). **Bowling**: like in real-life bowling, the agent must throw a bowling ball at pins, while ideally curving the ball so that it can clear all pins in one throw. The agent experiences rewards with a high delay, at the end of all rolls (between 2 and 4 depending on the number of strikes achieved). **Venture**: the agent must enter a room, collect a treasure and shoot monsters. Shooting monsters only give rewards after the treasure was collected, and there is no in-game reward for collecting it. **Seaquest**: the agent controls a submarine and must sink enemy submarines. To reach higher scores, the agent has to additionally rescue divers that only provide reward once the submarine lacks oxygen and surfaces to replenish it. **Solaris**: the agent controls a spaceship that earns points by hunting enemy spaceships. These shooting phases are followed by the choice of the next zone to explore on a high-level map, which conditions future rewards. **Skiing**: the agent controls a skier who has to go between poles while going down the slope. The agent gets no reward until reaching the bottom of the slope, at which time it receives a reward proportional to the pairs of poles it went through, which makes for long-term credit assignment.

VizDoom. VizDoom (Kempka et al., 2016) is a suite of partially observable 3D tasks based on the classical Doom video game, a first-person shooter. As mentioned before, it is an interesting sandbox for credit assignment because it optionally provides high-level information such as labelled game objects, depth as well as a top-view minimap representation; all of which can be used for approximate optimally efficient credit assignment algorithms.

BoxWorld. BoxWorld (Zambaldi et al., 2018) is a family of environments that shares similarities with Triggers, while being more challenging. Environments are also procedurally-generated square-shaped 2D gridworlds with discrete controls. The goal is to reach a gem, which requires going through a series of boxes protected by locks that can only be opened with keys of the same colour while avoiding distractor boxes. The relations between keys and locks can be utilised to assess assigned credit since the completion of the task (as well as intermediate rewards for opening locks) depends on the collection of the right keys.

Sokoban. Sokoban (Racanière et al., 2017) is a family of environments that is similar to the two previous ones. The agent must push boxes to intended positions on the grid while avoiding dead-end situations (for instance, if a block is stuck against walls on two sides, it cannot be moved anymore). While there is no definite criterion to identify decisive actions, actions that lead to dead-ends are known and can be exploited to assess the quality of credit assignment.

DeepMind Lab. DeepMind Lab (Beattie et al., 2016) (DMLab) is a suite of partially observable 3D tasks with rich visual input. We identify several tasks that might be of interest to assess credit assignment capabilities, some of which were used in recent work. **Keys-Doors**: the agent navigates to keys that open doors (identified by their shared colour) so that it can get to an absorbing state represented by a cake. Ferret et al. (2021a) consider a harder variant of the task where collecting keys is not directly rewarded anymore and feedback is delayed until opening doors. **Keys-Apples-Doors**: Hung et al. (2019) consider an extended version of the previous task. The agent still has to collect a key, but after a fixed duration a distractor phase begins in which it can only collect small rewards from apples, and finally, the agent must find and open a door with the key it got in the initial phase. To solve the task, the agent has to learn the correlation or causation link between the key and the door, which is made hard because of the extended temporal distance between the two events and of the distractor phase. **Deferred Effects**: the agent navigates between two rooms, the first one of which contains apples that give low rewards, while the other contains cakes that give high rewards but it is entirely in the dark. The agent can turn the light on by reaching the switch in the first room, but it gets an immediate negative reward for it. In the end, the most successful policy is to activate the switch regardless of the immediate cost so that a maximum number of cakes can be collected in the second room before the time limit.

7.3 Protocol

Online evaluation. The most standard approach is to evaluate the quality of credit assignment methods and individual credit assignments along the RL training procedure. As the policy changes, the credit assignments change since the effect of actions depends on subsequent actions (which are dictated by the policy). One can dynamically track the quality of credit assignments and that of the credit assignment method using the metrics developed in the previous section. For the credit assignment method, since it requires a dataset of interaction, one can consider using the most trajectories produced by the agent. An advantage of this approach is that it allows evaluating the evolution of the credit assignment quality along the RL training, with an evolving policy and resulting dynamics. Also, since the goal of credit assignment is to help turn feedback into improvements, it makes sense to evaluate it in the context of said improvements. While natural, online evaluation means one has little control over the data distribution of the evaluation. This is problematic because it is generally hard to disentangle credit quality from the nature of the trajectories it is evaluated on. A corollary is that outcomes that necessitate precise exploration (which can be the outcomes for which agents would benefit most from accurate credit assignment) might not be explored.

Offline evaluation. An alternative is to consider offline evaluation. It requires a dataset of interactions, either collected before or during the RL training. Credit assignments and the credit assignment method then use the parameters learned during the RL training while being evaluated on the offline data. As the policy in the offline data is generally not the latest policy from the online training, offline evaluation is better suited for policy-conditioned credit assignment or (to some extent) trajectory-conditioned credit assignment. Indeed, other forms of credit assignment are specific to a single policy, and evaluating these on data generated from another policy would not be accurate. An important advantage of offline evaluation is that it alleviates the impact of exploration, as one controls the data distribution credit is evaluated on.

However, it is not always possible to collect a dataset of interactions, and it is generally hard to collect a dataset that is representative of the full distribution of interactions the agent will face. This is especially true for tasks with high-dimensional observations, where the state space is vast and the agent can only explore a small fraction of it. In such cases, the offline evaluation might not be representative of the full distribution of interactions, and the evaluation might be biased. As a result, offline evaluation might not always be suitable to evaluate the quality of credit assignment methods, and a biased set of demonstration will lead to biased evaluations.

Relationship between online and offline evaluation. Since offline and online evaluations are complementary, it is often beneficial to use both. However, a natural question that arises is whether offline metrics correlate to online performance. While offline demonstrations can provide a more controlled and principled evaluation, disentangling credit assignment quality from exploration, it is crucial to verify that improvements measured offline translate to online gains. As discussed in the previous paragraphs, the correlation between offline and online metrics can only be guaranteed if the offline dataset is representative of the full distribution of interactions the agent will face. Indeed, the strength of this correlation may vary depending on the representativeness of the offline dataset: if critical or rare states are underrepresented, offline metrics could overestimate or underestimate the true online performance. For example, when evaluating the ability of an agent to play a videogame, a set of expert – even if sub-optimal – demonstrations might be a very good proxy for the full distribution of interactions, as it is likely to cover most of the states the agent will face.

For these reasons, if we can guarantee that the offline dataset is representative enough of the full distribution of interactions, offline evaluation can be a good proxy for online effectiveness. To conclude, while offline evaluation often proves useful and cost-effective, its results should be interpreted with caution and, ideally, supplemented by select online experiments to validate the offline-to-online correlation in practice.

Chapter 8

Closing, discussion and open challenges on Part III

The CAP is the problem to approximate the influence of an action from a finite amount of experience, and it is of critical importance to deploy RL agents into the real world that are effective, general, safe and interpretable. However, there is a misalignment in the current literature on what credit means in words and how it is formalised. In this chapters, we put the basis to reconcile this gap by reviewing the state of the art of the temporal CAP in Deep RL, focusing on three major questions.

8.1 Summary

Overall, we observed three major fronts of development around the CAP.

The first concern is the problem of *how to quantify action influence* (**Question 1.1.**). We addressed **Question 1.1.** in **Section 4**, and analysed the quantities that existing works use to represent the influence of an action. In **Section 4.1** we unified these measures of action influence with the *assignment* definition. In Sections 4.3 and 4.6 we showed that the existing literature agrees on an intuition of credit as a measure of the influence of an action over an outcome, but that it does not translate that well into mathematics and none of the current quantities align with the purpose. As a consequence, we proposed a set of principles that we suggest a measure of action influence should respect to represent credit.

The second front aims to address the question of *how to learn action influence from experience* and to describe the existing *methods* to assign credit. In **Section 5** we looked at the challenges that arise from learning these measures of action influence and, together with **Section 6**, answered **Question 1.2.**. We first reviewed the most common obstacles to learning already identified in the literature and realigned them to our newly developed formalism. We identified three dimensions of a POMDP, depth, breadth, and density and described pathological conditions on each of them that hinder the CA. In Section 6 we defined a CA method as an algorithm whose aim is to approximate a measure of action influence from a finite amount of experience. We categorised methods into those that: *(i)* use temporal contiguity as a proxy for causal influence; *(ii)* decompose the total return into smaller per-timestep contributions; *(iii)* condition the present on information about the future using the idea of hindsight; *(iv)* use sequence modelling and represent action influence as the likelihood of action to follow a state and predict an outcome; *(v)* learn to imagine backward transitions that always start at a key state and propagate back to the state that could generate them; *(vi)* meta-learn action influence measures.

Finally, the third research front deals with *how to evaluate quantities and methods* to assign credit and aims to provide an unbiased estimation of the progress in the field. In **Section 7** we addressed **Question 1.3.** and analysed how current methods evaluate their performance and how we can monitor future advancements. We discussed the resources that each benchmark has to offer and their limitations. For example, diagnostic benchmarks do not isolate the specific CAP challenges identified in Section 5: delayed effects, transpositions, and sparsity. Benchmarks at scale often cannot disentangle the CAP from the exploration problem, and it becomes hard to understand whether a method is advancing one problem or another.

8.2 Discussion and open challenges to date

As this chapters suggests, the work in the field is now fervent and the number of studies in a bullish trend, with many works showing substantial gains in control problems only by – to the best of our current knowledge – advancing on the CAP alone (Bellemare et al., 2017; van Hasselt et al., 2021; Edwards et al., 2018; Mesnard et al., 2021; 2023).

We observed that the take-off of CA research in the broader area of RL research is only recent. The most probable reason for this is to be found in the fact that the tasks considered in earlier Deep RL research were explicitly designed to be simple from the CA point of view. Using tasks where assigning credit is hard would have – and probably still does, e.g., Küttler et al. (2020) – obfuscate other problems that it was necessary to solve before solving the CAP. For example, adding the CAP on the top of scaling RL to high-dimensional observations (Arulkumaran et al., 2017) or dealing with large action spaces (Dulac-Arnold et al., 2015; van Hasselt & Wiering, 2009) would have, most likely, concealed any evidence of progress for the underlying challenges. This is also why CA methods do not usually shine in classical benchmarks (Bellemare et al., 2013), and peer reviews are often hard on these works. Today, thanks to the advancements in other areas of RL, the field is in a state where improving on the CAP is a compelling challenge.

Yet, the CAP still holds open questions and there is still much discussion required to consider the problem solved. In particular, the following observations describe our positions with respect to this survey.

Aligning future works to a common problem definition. The lack of a review since its conception (Minsky, 1961) and the rapid advancements produced a fragmented landscape of definitions for action influence, an ambiguity in the meaning of *credit assignment*, a misalignment between the general intuition and its practical quantification, and a general lack of coherence in the principal directions of the works. While this diversity is beneficial for the diversification of the research, it is also detrimental to comparing the methods. Future works aiming to propose a new CA method should clarify these preliminary concepts. Answers to "What is the choice of the measure of action influence? Why the choice? What is the method of learning it from experience? How is it evaluated?" would be good a starting point.

Characterising credit. "*What is the* minimum *set of properties that a measure of action influence should respect to inform control? What the more desirable ones?*". This question remains unanswered, with some ideas in Ferret (2022, Chapter 4), and we still need to understand what characterises a proper measure of credit.

Causality. The relationship between CA and causality is underexplored, but in a small subset of works (Mesnard et al., 2021; Pitis et al., 2020; Buesing et al., 2019). The literature lacks a clear and complete formalism that casts the CAP as a problem of causal discovery. Investigating this connection and formalising a measure of action influence that is also a satisfactory measure of causal influence would help better understand the effects of choosing a measure of action influence over another. Overall, we need to better understand the connections between CA and causality: what happens when credit is a strict measure of causal influence? How do current algorithms perform with respect to this measure? Can we devise an algorithm that exploits a causal measure of influence?

Optimal credit. Many works refer to *optimal credit* or to *assigning credit optimally*, but it is unclear what that formally means. "*When is credit optimal?*" remains unanswered.

Combining benefits from different methods. Methods conditioning on the future currently show superior results compared to methods in other categories. These promising methods include hindsight (Section 6.4), sequence modelling (Section 6.5) and backward learning and planning methods (Section 6.6). However, while hindsight methods are advancing fast, sequence modelling and backward planning methods are underinvestigated. We need a better understanding of the connection between these two worlds, which could potentially lead to even better ways of assigning credit. Could there be a connection between these methods? What are the effects of combining backward planning methods with more satisfactory measures of influence, for example, with CCA?

Benchmarking. The benchmarks currently used to review a CA method (Chevalier-Boisvert et al., 2018; Bellemare et al., 2013; Samvelyan et al., 2021) (see Section 7.2) are often borrowed from *control* problems, leading to the issues discussed in Section 7 and recalled in the summary above. On a complementary note, CA methods are often evaluated in actor-critic settings (Harutyunyan et al., 2019; Mesnard et al., 2021), which adds layers of complexity that are not necessary. This, together with the inclusion of other unnecessary accessories, can obfuscate the contributions of CA to the overall RL success. As a consequence, the literature lacks a fair comparison among all the methods, and it is not clear how all the methods in Section 6 behave with respect to each other against the same set of benchmarks. This lack of understanding of the state of the art leads to a poor signal to direct future research. We call for a new, community-driven single set of benchmarks that disentangles the CAP from the exploration problem and isolate the challenges described in Section 5. How to disentangle the CAP and the exploration problem? How to isolate each challenge? Shall we evaluate in value-based settings, and would the ranking between the methods be consistent with an evaluation in actor-critic settings? While we introduced some ideas in Section 5.5, these questions are still unanswered.

Reproducibility. Many works propose open-source code, but experiments are often not reproducible, their code is hard to read, hard to run and hard to understand. Making code public is not enough, and cannot be considered open-source if it is not easily usable. Other than public, open-source code should be accessible, documented, easy to run, and accompanied by continuous support for questions and issues that may arise from its later usage. We need future research to acquire more rigour in the way to publish, present, and support the code that accompanies scientific publications. In particular, we need *(i)* a formalised, shared and broadly agreed standard that is not necessarily a *new* standard; *(ii)* for new studies to adhere to this standard, and *(iii)* for publishers to review the accompanying code at least as thoroughly as when reviewing scientific manuscripts.

Monitoring advancements. The community lacks a database containing comprehensive, curated results of each baseline. Currently, baselines are often re-run when a new method is proposed. This can potentially lead to comparisons that are unfair both because the baselines could be suboptimal (e.g., in the hyperparameters choice, training regime) and their reproduction could be not faithful (e.g., in translating the mathematics into code). When these conditions are not met, it is not clear whether a new method is advancing the field because it assigns credit better or because of misaligned baselines. We call for a new, community-driven database holding the latest evaluations of each baseline. The evaluation should be driven by the authors and the authors be responsible for its results. When such a database will be available, new publications should be tested against the same benchmarks and not re-run previous baselines, but rather refer to the curated results stored in the database.

Peer reviewing CA works. As a consequence of the issues identified above, and because CA methods do not usually shine in classical benchmarks (Bellemare et al., 2013), peer reviews often do not have the tools to capture the novelties of a method and its improvements. On one hand, we need a clear evaluation protocol, including a shared benchmark and leaderboard to facilitate peer reviews. On the other hand, peer reviews must steer away from using tools and metrics that would be used for control, and use those appropriate for the CAP instead.

Lack of priors and foundation models. Most of the CA methods start to learn credit from scratch, without any prior knowledge but the one held by the initialisation pattern of its underlying network. This represents a main obstacle to making CA efficient because, at each new learning phase, even elementary associations must be learned from scratch. In contrast, when facing a new task, humans often rely on their prior knowledge to determine the influence of an action. In the current state of the art, the use of priors to assign credit more efficiently is overlooked. Vice versa, the relevance of the CAP and the use of more advanced methods for CA (Mesnard et al., 2021; 2023; Edwards et al., 2018; van Hasselt et al., 2021) is often underestimated for the development of foundation models in RL.

8.3 Conclusions

To conclude this part, in this **Part III**, we have set out to formally settle the CAP in Deep RL. The resulting material does not aim to solve the CAP, but rather proposes a unifying framework that enables a fair comparison among the methods that assign credit and organises existing material to expedite the starting stages of new studies. Where the literature lacks answers, we identify the gaps and organise them in a list of challenges.

Part IV

Advancing the Credit Assignment Problem

Chapter 9

Credit Assignment with Language Models

Canonical solutions to the CAP, such as *reward shaping* and *options*, require extensive domain knowledge and manual intervention, limiting their scalability and applicability. In this work, we lay the foundations for CALM, a novel approach that leverages LLMs to automate credit assignment via reward shaping and options discovery. CALM uses LLMs to decompose a task into elementary subgoals and assess the achievement of these subgoals in state-action transitions. Every time an option terminates, a subgoal is achieved, and Credit Assignment with Language Models (CALM) provides an auxiliary reward. This additional reward signal can enhance the learning process when the task reward is sparse and delayed without the need for human-designed rewards. We provide a preliminary evaluation of CALM using a dataset of human-annotated demonstrations from MiniHack, suggesting that LLMs can be effective in assigning credit in zero-shot settings, without examples or LLM fine-tuning. Our preliminary results indicate that the knowledge of LLMs is a promising prior for credit assignment in RL, facilitating the transfer of human knowledge into value functions.

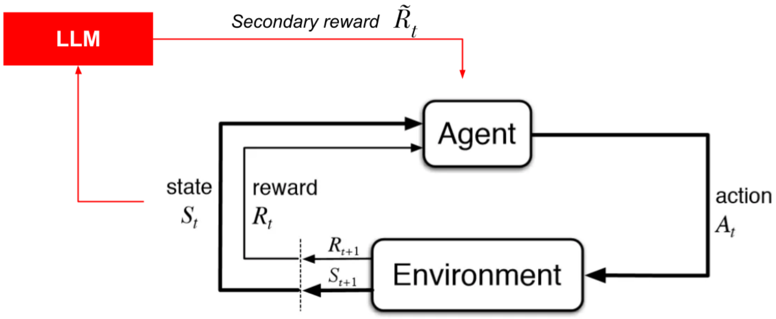


Figure 9.1: A schematic representation of the CALM method. The LLM is used to evaluate the actions of an RL agent in a POMDP environment. The LLM is provided with a description of the task and a transition, and is asked to determine if the action taken in the transition makes progress towards solving the task. The LLM is used as a critic to assign credit to the agent's actions, providing an additional reward signal to enhance the learning process.

9.1 Introduction

We established in the **Part III** how and why the CAP (Minsky, 1961; Sutton, 1984; Pignatelli et al., 2024a) is a fundamental challenge in RL. It typically involves determining the contribution of each action to the final outcome, a process crucial for accurate policy evaluation. Effective CA enables agents to learn useful associations between actions and outcomes, and provides useful directions to improve the policy.

However, when rewards are dispensed only at the end of a task (Efroni et al., 2021), as it is often the case, the feedback becomes sparse and delayed, making CA particularly challenging. In such scenarios, rewarding events are rare, and Deep RL agents often struggle to convert occasional successes into a robust decision-making process. To exacerbate the issue, RL agents typically begin with no prior knowledge (*tabula rasa*) and must learn the nuances and intricacies of complex tasks from scratch. The lack of controlled experimental conditions, such as the ability to observe counterfactuals, makes it difficult for them to distinguish between correlation and causation. As a result, tasks that are usually easy to solve for humans become hard to address for an RL agent.

To address these challenges, many methods incorporate prior human knowledge into RL systems. Two techniques are canon: reward shaping (Ng et al., 1999; Gupta et al., 2022) and Hierarchical Reinforcement Learning (HRL) (Al-Emran, 2015; Sutton et al., 1999) via options (Sutton et al., 1999). Reward shaping involves providing an additional synthetic reward to guide the agent's actions when natural rewards are uninformative. HRL decomposes complex tasks into simpler ones (*options*), training agents to achieve intermediate objectives that provide a signal while the MDP would not. Despite their effectiveness, these methods require extensive human input, making them costly and difficult to scale across different environments.

Recently, LLMs have emerged as a useful tool to transfer human knowledge into computational agents, either through planning (Dalal et al., 2024), expressing preferences (Klissarov et al., 2023), or grounding their abstract knowledge into practical solutions (Huang et al., 2023; Carta et al., 2023). Notably, these models have produced strong results in causal reasoning tasks (Jin et al., 2023) with performances comparable to humans (Kıcıman et al., 2023). These results suggest that LLMs could be an effective, supplementary tool to distinguish between correlation and causation more effectively than traditional methods used in early stages of RL training.

With these results, a natural question arises: "*Can the knowledge encoded in LLMs serve as a useful prior for CA in RL?*" Inspired by the successes of LLMs, we introduce CALM, a general method to perform CA with LLMs using reward shaping. We hypothesize that the prior knowledge of a LLM can provide valuable signals that improve CA in RL, and propose a way to transfer these priors into the agent's value function. On this assumption, CALM leverages a pretrained LLM to break down tasks into smaller, composable subgoals and determine if a state-action-state transition achieves a subgoal. This provides an additional reward signal to enhance RL algorithms, and effectively automates reward shaping by substantially reducing the involvement of humans in the training loop.

We present a preliminary evaluation of the efficacy of CALM in zero-shot settings, with no examples and no finetuning. We collect a dataset of demonstrations from MiniHack (Samvelyan et al., 2021) and use it to compare the performance of LLMs against human annotations. Our results indicate that LLMs are a viable means to transfer common human knowledge into value functions, and can be effective in automating reward shaping. This bodes well for the prospect to improve CA in the full RL problem.

9.2 Related work

LLMs for RL. Recent advancements have shown the potential of pretrained LLMs in enhancing RL agents. Paischer et al. (2022; 2024) used CLIP encodings to improve the state representations of POMDPs. Yao et al. (2020); Du et al. (2023) investigated the ability of pretrained LLMs to improve exploration. Huang et al. (2023); Carta et al. (2023) grounded the abstract knowledge of these models and their capabilities into practical RL tasks. LLMs have been used for planning, either directly as world models (Huang et al., 2022; Wang et al., 2023; Singh et al., 2023; Brohan et al., 2023; Dasgupta et al., 2023; Shah et al., 2023; Zhong et al., 2020; 2022) or by writing code (Liang et al., 2022). Unlike these methods we use pretrained LLMs as a critic: the LLM provides an *evaluation* of an action for how useful it is to achieve a goal in the future. Among the methods above, Du et al. (2023) is the only method to use subgoals, but these are used to condition a goal-oriented policy, rather than as a critic.

LLMs for reward shaping. Carta et al. (2022); Goyal et al. (2019b) explore the advantages of using pure language abstractions for reward shaping, but do not use a pretrained LLMs and its prior knowledge. Kwon et al. (2023) use the responses of LLMs as a reward signal, but the investigation is limited to conversational environments.

LLMs for knowledge transfer. Another set of studies used intrinsic rewards to transfer the prior knowledge of an LLM to a value function. Wu et al. (2024) used LLMs to provide an auxiliary reward signal in Atari (Bellemare et al., 2013), based on the information contained in a game manual. Unlike this study, we use subgoals to extract the reward signal, and we do not focus on incorporating external knowledge material, but rely on the LLM's prior knowledge to solve the task. Klissarov et al. (2023) constructed a reward function from the LLM's preferences over NetHack (Küttler et al., 2020) in-game messages only. Instead, our method incorporates the full observation, does not use preferences, and does not require a separate stage to fit the preference set, but uses the LLM's output directly.

In short, none of these methods proposes to generalise reward shaping with hierarchical skills using pretrained LLMs. Unlike the methods above, we use pretrained LLMs as a critic: we aim to uncover cause-effect relationships between actions and goals by both breaking down a task into valuable subgoals and then acting as a reward function for them. This provides an intermediate signal to shape the agent's behaviour when rewards are sparse and delayed.

9.3 Assumptions

We consider the problem of learning to solve POMDPs, as described in **Chapter 2**.

To best isolate the CAP from other problems, we focus only on environments with a discrete action space, and deterministic state transitions. To evaluate the capabilities of LLMs in environments where the CAP is hard, we only consider tasks where the reward signal is delayed. Specifically, the reward function is 0 everywhere, and 1 when a goal state is reached.

To start the investigation, we evaluate the LLM only in language settings, leaving the investigation of multimodal ones (text, image, audio, video) to future works. For this reason, we consider only environments with an observation kernel that maps states to a textual codomain, $O : \mathcal{S} \to \mathcal{T}$, where $\mathcal{T}$ is a set of sequences of characters.

When comparing the LLM's performance with human annotations, we assume that these annotations are optimal. However, due to the natural breadth (see Section 5.3) of the POMDP, these annotations might not me the only optimal solution. For example, the LLM might break down a task into different subgoals that are equally valid, but not exactly the same. For this reason, our evaluation is limited to *alignment* with human annotations, rather than optimality.

Finally, we consider a black box, pretrained LLM, that takes an input text and maps it to a finite set of output characters. We consider only open-weights models that can fit an NVIDIA A100 80Gb in either 16 bits floating point or 4 bits integer mode. We assume that the LLM has enough preliminary knowledge of the MiniHack environment to recognise valuable actions that progress towards a win.

9.4 Methods

We set out to design a general method to assign credit in RL using LLMs that can generalise to multiple tasks with little human input. Next, we formalise the method, discuss its assumptions and provide details on the protocols we use to evaluate it.

9.4.1 Reward shaping

Among the available CA techniques, we focus on *reward shaping* (Ng et al., 1999), due to both its effectiveness in assigning credit and its limitations to generalisation related to the costs of human involvement in the training loop. Reward shaping aims to address the scarcity of learning signals by introducing an auxiliary reward function, the *shaping function*:

$$\tilde{r}_{t+1} = \tilde{R}(s_t, a_t, s_{t+1}). \tag{9.1}$$

Here, s_t is the state at time t, a_t is the action taken in that state, s_{t+1} is the resulting state, and $\tilde{r}_{t+1}$ is the auxiliary reward collected after taking a_t in s_t. This reward is added to the original reward signal $R(s_t, a_t, s_{t+1})$ to obtain the new, shaped reward

$$r_{t+1} = R(s_t, a_t, s_{t+1}) + \tilde{R}(s_t, a_t, s_{t+1}). \tag{9.2}$$

If there exist a function $\phi : \mathcal{S} \to \mathbb{R}$ such that $\tilde{R}(s_t, a_t, s_{t+1}) = \phi(s_{t+1}) - \phi(s_t)$, then the set of optimal policies is preserved, and the shaping function is also a *potential function* (Ng et al., 1999).

While potential-shaping can be proven in theory for MDPs, we consider the more general problem of solving POMDPs. Assuming that we do not have access to the true, underlying MDP state, these theoretical guarantees do not hold. For this reason, in the following, we consider the more general case of non-optimality preserving functions, and assume that the set of optimal policies induced by the shaped reward is good enough for the agent to satisfatcory solve the task.

For example, in key-door environments, a common testbed for CA methods (Hung et al., 2019; Mesnard et al., 2021), the agent must reach a reward behind a locked door, which can only be opened if the agent possesses a key. Here, the agent has clear subgoals: (*i*) go to the key, (*ii*) pick it up, (*iii*) go to the door, (*iv*) unlock it, (*v*) go to the reward. Achieving these subgoals sequentially leads to optimal behaviour. However, the agent struggles to recognise this hierarchical pattern due to the lack of immediate feedback from the environment. This is particularly true in the early stages of training, when behaviour is erratic, and two optimal actions can be separated by a long sequence of random ones. Providing intermediate feedback for each achievement often improves the agent's performance (Gupta et al., 2022), and the ability of $\tilde{R}$ to produce an instantaneous signal indicating progress is crucial for better CA. Thus, reward shaping can significantly accelerate the learning process in environments with sparse or delayed rewards.

However, designing an effective shaping function is challenging. The function should be carefully designed to provide useful guidance without leading to unintended behaviours. This often calls for incorporating domain knowledge or heuristic information about the task, and requires deep task and environment knowledge. Such knowledge may not be readily available or easily codifiable, limiting the applicability of reward shaping in diverse or unknown environments. This process is complex and time-consuming, and it might not always be possible to devise a reward function that incentivizes learning, is computationally cheap, and general enough to adapt to various tasks. Improving this limitation could enable broader use of reward shaping and enhance CA in deep RL.

9.4.2 LLMs as shaping functions

Encouraged by the recent successes of LLMs in RL (Klissarov et al., 2023) and of using language to abstract skills (Jiang et al., 2019; Jacob et al., 2021; Sharma et al., 2021; Mu et al., 2022), we explore whether these models can offer a valid alternative to humans in the reward shaping process. Our goal is to produce a function that, given a description of the task and a state-action-state transition, produces a binary signal indicating whether the action makes progress towards solving the task or not:

$$LLM : desc(\mathcal{M}) \times desc(\mathcal{S} \times \mathcal{A} \times \mathcal{S}) \to \mathbb{B}. \tag{9.3}$$

Here, LLM is a pretrained LLM; $desc(\mathcal{M})$ is a natural language description of the POMDP (the task); $desc(\mathcal{S} \times \mathcal{A} \times \mathcal{S})$ is a textual representation of the transition, not necessarily in natural language (for example, a grid-arranged text), and $\mathbb{B} = \{0, 1\}$ is the Boolean domain. In this scenario, the LLM acts as a critic: its role is to *evaluate* the action a_t in the transition (s_t, a_t, s_{t+1}) based on the heuristics that we describe next.

We operationalise the idea using the notion of *options* (Sutton et al., 1999): An *option* is a temporally extended actions that often represent abstractions over the action space of the POMDP, and consists of two elements: an intra-option policy $\pi_i : \mathcal{S} \rightarrow \Delta(\mathcal{A})$, and a termination condition $\beta : \mathcal{S} \rightarrow \mathbb{B}$.[1] Indeed, to develop an intuition of options, it is useful to visualise one as a macro-action: a set of actions that, taken together, have precise semantics. For example, in our key-to-door example, one useful option to consider is to *pick up the key*. This macro action includes a set of primitive actions – the set of actions to navigate to the key and the action *pickup* – and a termination condition – whether the key is picked up. For the purpose of our analysis, this termination is crucial, as it signals that the subtask has been successfully achieved.

We exploit this idea to build our shaping function, set up a single-turn conversation, and prompt the model to perform two subtasks:

(i) To identify a set of useful options in the environment, by breaking down the task into a sequence of shorter subgoals. These options, and more specifically their termination, effectively constitutes our set of subgoals, since a subgoal is achieved when the option terminates (a key is picked up).

(ii) Determine whether an option terminated (thus, if a subgoal is achieved) in the transition (s_t, a_t, s_{t+1}).

[1] We consider $\mathcal{S}^+ = \mathcal{S}$ and omit the initiation set $\mathcal{S}^+$

Every time an option terminates, we augment the task reward with the subtask reward as according to our reward shaping rule, $\tilde{R}(s_t, a_t, s_{t+1}) = \beta(s_{t+1})$.

This way of segmenting the task into subgoals is an alternative implementation of the option framework – compared to the traditional one that specifies an intra-option policy and a policy over options – but a common one (Sutton et al., 2011; Schaul et al., 2015a). In the broader context of CA methods, many approaches related to options are a solid and promising way to improve on the CAP, as discussed in Section 8. However, most of them fall short in providing a general method to discover subgoals. Options are pretedermined, must be provided *ex-ante* and require extensive domain knowledge to be effective. For this reason, by levereging the general knowledge of an LLM, CALM provides a way to define goal-conditioned influence function (Pignatelli et al., 2024a) with autonomously discovered subgoals, that can be used to assign credit in a wide range of tasks, without the need for human intervention.

In essence, Equation (9.3) aims to mimic a human supervising an RL agent's decisions, acting as an auxiliary critic. Decomposing the task into multiple subgoals allows each sub-achievement to correspond to a small step towards success, and composing all the subgoals sequentially results in successful behaviour. Since achieving a subgoal is contingent on achieving all the preceding ones, the number of subgoals achieved quantifies the agent's progresses. To develop an intuition of the idea, subgoals can be thought of as levels; gaining a level at the current time indicates progress in achieving a specific goal in the future. This process of *actualisation*, where an action is evaluated for its future potential to achieve a goal, characterises the function as a CA method (Pignatelli et al., 2024a).

9.5 Experimental protocol

The viability of CALM in online RL settings depends on the quality of the assignments provided by the LLM. Good quality assignments – signals that reinforce optimal actions – can improve the performance of an RL algorithm. Thus, we provide a preliminary evaluation of CALM on an offline dataset of demonstrations.

Environment. We focus on the KeyRoom environment, a canonical testbed for CA methods (Hung et al., 2019; Mesnard et al., 2021; 2023) originally proposed in Minigrid (Chevalier-Boisvert et al., 2018). We choose its MiniHack version, for it provides a textual representation of the observations that can be fed to a language system. The game presents a top-down view of a grid-like environment with two rooms. The agent starts in the first room, where a key is located. It must pick up the key and use it to unlock the door to the second room, where a reward is located. We consider two types of observations:

1. **Cropped observations.** a top-down, north-facing, 9x9 crop around the agent, which is known to improve the performance in standard RL benchmarks on Nethack (Küttler et al., 2020).

2. **Game screens.** A top-down, north-facing, 21x79 grid showing the entire game scene, including an in-game message and a set of statistics of the current state. We also refer to these as *human* observations, since they reproduce the conditions of human game play.

Both observations are partial, despite containing different amounts of information. We consider a discrete action set: *go north*, *go east*, *go south*, *go west*, *pickup*, *apply*. The reward function is deterministic, providing a reward of 1 if the agent reaches the goal tile and 0 otherwise. Transitions are also deterministic.

Dataset. We collect 256 one-step transitions $d_t = (s_t, a_t, s_{t+1})$ using a random policy. Given a set of subgoals $\mathcal{G} \subset (\mathcal{S} \times \mathcal{A} \times \mathcal{S})$, a transition d_t can then be classified as either achieving a subgoal $g \in \mathcal{G}$ or not. This produces categories $\mathcal{C} = \{c_i \,:\, 0 \leq i \leq |\mathcal{G}| + 1\}$, one for each subgoal, and an additional one when no subgoal is achieved. To characterise the abilities of an LLM to assign credit accurately, we produce a balanced dataset where each goal appears with equal probability.

Composing the prompt. For each transition we then compose a prompt using the following structure:

1. `<ROLE>` specifies the role we ask the LLM to simulate.
2. `<ENVIRONMENT-DESCRIPTION>` describes the RL environment, the source of the gameplay.
3. `<SYMSET>` is a list reporting Nethack wiki entries[2] of what each symbols in the grid represents.
4. `<TASK-DESCRIPTION>` specifies the overall goal of the agent, and does not contain information about subgoals.
5. `<SUBGOALS>` contains either a list of subgoals to achieve, or asks the LLM to produce one.
6. `<INSTRUCTIONS>` tasks the agent to determine whether a subgoal is achieved in the trajectory presented in `<TRANSITION>`.
7. Finally, `<OUTPUT-FORMAT-REQUEST>` requests the output in a format that can be easily parsed, for example, a python dictionary.

Prompt 1 shows a concrete instantiation of this structure, where goals are provided as part of the input. Here, the role is not specified, exhorting the

[2] https://nethackwiki.com/wiki/Symset

LLM to play a generic role, and the environment description (*The environment is MiniHack*) is minimal. In the symset – the list of symbols with their meaning – the descriptions are extracted from the wiki (`https://nethackwiki.com/wiki/Symset`). The task is as generic as possible (*to win the game*), and it is followed by the set of predetermined subgoals (*pick up the key* and *unlock the door*). The instructions and the request for an appropriate output format follow on that. Finally, we enclose the transition within a `<gameplay>` tag, and remark that this is a single-turn conversation to avoid the model asking additional clarifications. Notice that we separate each cell in the observation with a whitespace to ensure that each cell (plus their whitespace) corresponds to a separate token. We discuss this more in depth in Appendix A.6.1, and provide more details and variations of prompts in Appendix A.3. To develop an intuition of the role covered by the model, we encourage the reader to scan over them before proceeding.

Models. We use pretrained, open-weights large language models that can fit a 80Gb A100 Nvidia GPU in either 16 bits brain floating point (Dean et al., 2012) or 4 bits integer weights representations. When models cannot fit in memory, we use their NF4 (Dettmers et al., 2023) quantised equivalent. These models are marked with an asterisk (*) in the tables below. All the models are finetuned for instructions following, and tokens are deterministically sampled using a greedy policy.

Annotations. For each transition a human annotator produces a term of reference for comparison. The annotator is presented with each prompt in the dataset, without any further instructions. We then record the annotator's answer, and use it as a term of reference for the LLM's responses. Since the prompt has a correct answer, these are not subjective evaluations, but rather direct verification, with little room for interpretation.

Example prompt

```
The environment is MiniHack.

I will present you with a short extract of a gameplay. At each timestep, symbols represent the following items:
- "." represents a floor tile.
- "|" can represent either a wall, a vertical wall, an open door.
- "-" can represent either the bottom left corner (of a room), bottom right corner (of a room), wall, horizontal wall,
wall, top left corner (of a room), op right corner (of a room).
- "+" represents a closed door. Doors can be locked, and require a key to open.
- "(" represents a useful item (pick-axe, key, lamp...)
- "<" represents a ladder or staircase up.
- ">" represents a ladder or staircase down.

The task of the agent is to win the game.

First, based on your knowledge of NetHack, break down the task of the agent into subgoals.
Then, consider the following game transition, which might or might not contain these subgoals.
Determine if any of the subgoals is achieved at Time: 1 or not.


Report your response in a dictionary containing the name of the subgoals as keys and booleans as value. For example:
‘‘‘python
{
    <name of goal>: <bool>,
}

Observation Sequence:

<gameplay>
Time: 0
Current message:

      - - - -
      | . . |
      | . . |
- - + - . < |
| . . . @ . |
| . ( . . . |
- - - - - - -


Time: 1
Current message:

      | . . |
      | . . |
- - + - . < |
| . . . . . |
| . ( . @ . |
- - - - - - -



</gameplay>

I will not consider anything that is not in the dictionary.
You have only one shot at this, and you cannot ask for clarifications.
```

Prompt 1: Example of a prompt for instruction verification. Here, goals are provided externally from a human.

Evaluation. We then compare the LLM's annotations with the human ones. The response is a true positive if both the LLM and the human annotator identify that a subgoal is achieved. It is a false positive (a *hallucination*) if the LLM identified it, but the human has not; a false negative (a *miss*) if the human identified one, but the agent has not. This effectively casts the problem as classification, with the set of classes $\mathcal{C}$, as described in the dataset description. We then compare the LLM's hypotheses with the human responses as ground truth, and report accuracy, F1 score, precision and recall.

Notice that the aim of this experiments is not to evaluate whether the LLM is better than a human annotator with respect to an optimal solution. In fact, as also explained in Section 9.3, the POMDP might have more than one optimal solution, for example by breaking the task into different subgoals that are equally valid, but not exactly the same. Instead, recalling that our main objective is to investigate whether LLMs can serve as a proxy of human knowledge to automate reward shaping in RL, we focus on the *alignment* of the LLM's evaluation with the human's one.

9.6 Experiments, results, and discussion

To evaluate the effectiveness of LLMs in CA for RL, we consider environments with textual representations. We assume that the LLM has sufficient knowledge of the game to evaluate actions. While this assumption might be strong for NetHack, it is reasonable for MiniHack, where tasks are simplified yet challenging models of common NetHack scenarios, requiring only partial knowledge.

Based on the set of experimental conditions described above, we then consider a spectrum of settings requiring progressively less input from humans. We start by providing the LLM with: *(a)* cropped observations focused around the agent; *(b)* an effective, predetermined set of subgoals; We then proceed to progressively relax these conditions to: *(a)* gamescreen observations; *(b)* allowing the LLM to discover useful subgoals autonomously. These conditions are set to replicate the conditions of a human playing the game.

9.7 Can LLMs understand goal specifications and verify option termination?

This experiment aims to assess whether a pretrained LLM can function as a reward function when subgoals are provided externally. We provide the LLM with the environment name, `MiniHack`, and a list of two subgoals: `pick up the key` and `unlock the door`. We specify that the goal of the agent is simply to *win the game* (Jeurissen et al., 2024), and ask it to determine if each subgoal has been achieved in the transition. Prompt 4 shows an example prompt for this experiment.

We present results for multiple pretrained LLMs, using both cropped observations and full game screens. The purpose of the comparison is not to determine a winning model. It is, instead, to understand whether the ability to assign credit to single transitions is in the spectrum of capabilities of existing open-weights LLMs. This will lay the foundation for applying the method in full RL settings.

Annotator	F1 ↑	Accuracy ↑	Precision ↑	Recall ↑	TP ↑	TN ↑	FP ↓	FN ↓
Human	**1.00**	**1.00**	**1.00**	**1.00**	**171**	**85**	**0**	**0**
Mixtral-8x7B-Instruct-v0.1*	**0.74**	0.67	0.77	**0.73**	**124**	47	38	**47**
gemma-1.1-7b-it	0.73	**0.70**	0.91	0.61	105	75	10	66
Meta-Llama-3-70B-Instruct	0.66	0.65	**0.97**	0.50	85	82	3	86
Meta-Llama-3-8B-Instruct	0.64	0.64	0.95	0.49	83	81	4	88
c4ai-command-r-v01*	0.60	0.57	0.80	0.49	83	64	21	88
Mistral-7B-Instruct-v0.2	0.48	0.54	0.96	0.32	55	83	2	116
gemma-1.1-2b-it	0.00	0.33	0.00	0.00	0	**85**	**0**	171
Random	0.33	0.33	0.33	0.33				

Table 9.1: Performance of LLM annotations against human annotations with **game screen** observations and with the subgoals **provided** in the prompt. Models marked with an asterisk (*) are quantised to NF4 format. TP stands for *true positives*, TN for *true negatives*, FP for *false positives*, and FN for *false negatives.* Rows sorted by F1 score.

We report results in Tables 9.1 and 9.2, and draw the following two insights. First, LLMs, except *gemma-1.1-2b-it*, probably due to its small size, are generally effective in recognising when an instruction has been successfully com-

Annotator	F1 ↑	Accuracy ↑	Precision ↑	Recall ↑	TP ↑	TN ↑	FP ↓	FN ↓
Human	**1.00**	**1.00**	**1.00**	**1.00**	**171**	**85**	**0**	**0**
Mixtral-8x7B-Instruct-v0.1*	**0.78**	**0.70**	0.78	**0.77**	**132**	48	37	**39**
gemma-1.1-7b-it	0.76	0.69	0.79	0.73	124	52	33	47
gemma-1.1-2b-it	0.76	0.68	0.76	**0.77**	131	43	42	40
c4ai-command-r-v01*	0.75	0.69	0.81	0.70	120	57	28	51
Meta-Llama-3-70B-Instruct	0.63	0.58	0.76	0.54	92	56	29	79
Meta-Llama-3-8B-Instruct	0.61	0.61	0.92	0.46	79	78	7	92
Mistral-7B-Instruct-v0.2	0.61	0.62	**0.96**	0.45	77	**82**	**3**	94
Random	0.33	0.33	0.33	0.33				

Table 9.2: Performance with **cropped** observations and with the subgoals **provided** in the prompt.

pleted in a state-action-state transition. This shows their ability to understand goal specifications and to recognise when an option terminates due to completion. We also noticed that *c4ai-command-r-plus* degenerates into outputting *false* for most transitions, most probably due to quantisation.

Second, restricting the field of view of the observation helps improve performance. This is most likely due to observations being more concise, and avoiding the information to drown among a high number of tokens. This also seems to increase the lower bound, and the performance of models drastically failing with *human* observations greatly improves, especially *gemma-1.1-2b-it*.

9.7.1 Can LLMs suggest effective options?

In this experiment, we evaluate whether LLMs can autonomously suggest effective options. Instead of providing a predetermined list, we ask the LLM to break down the task into subgoals and verify whether these subgoals have been achieved. Despite only a small change on the surface, removing some key information from the prompt intensively tests the LLM's knowledge of NetHack. More importantly, it stresses the ability of the models to come up with a viable and effective hierarchy of subgoals such that, if reinforced, produces useful signals for progress.

This setting is more complex but also more general, as it replicates the amount of information typically available to a human player. Prompt 5 shows an example prompt for this experiment. As for the previous experiment, we evaluate the performance of different models using both cropped and human observations.

Annotator	F1 ↑	Accuracy ↑	Precision ↑	Recall ↑	TP ↑	TN ↑	FP ↓	FN ↓
Human	**1.00**	**1.00**	**1.00**	**1.00**	**171**	**85**	**0**	**0**
Meta-Llama-3-70B-Instruct	**0.82**	**0.72**	0.71	**0.96**	**165**	19	66	**6**
Meta-Llama-3-8B-Instruct	0.80	0.70	0.72	0.89	153	26	59	18
gemma-1.1-7b-it	0.77	0.66	0.71	0.85	145	25	60	26
Mixtral-8x7B-Instruct-v0.1*	0.74	0.64	0.71	0.76	130	33	52	41
Mistral-7B-Instruct-v0.2	0.57	0.48	0.63	0.53	90	32	53	81
c4ai-command-r-v01*	0.56	0.52	0.71	0.47	80	52	33	91
gemma-1.1-2b-it	0.00	0.33	0.00	0.00	0	**85**	**0**	171
Random	0.33	0.33	0.33	0.33				

Table 9.3: Performance with **game screen** observations and with **autonomously discovered** subgoals.

Annotator	F1 ↑	Accuracy ↑	Precision ↑	Recall ↑	TP ↑	TN ↑	FP ↓	FN ↓
Human	**1.00**	**1.00**	**1.00**	**1.00**	**171**	**85**	**0**	**0**
Meta-Llama-3-70B-Instruct	**0.83**	**0.75**	0.75	0.93	159	33	52	12
gemma-1.1-7b-it	0.81	0.70	0.71	**0.95**	**163**	17	68	**8**
Mixtral-8x7B-Instruct-v0.1*	0.72	0.62	0.71	0.74	127	32	53	44
Mistral-7B-Instruct-v0.2	0.65	0.54	0.66	0.64	109	28	57	62
c4ai-command-r-v01*	0.60	0.52	0.68	0.54	92	41	44	79
gemma-1.1-2b-it	0.47	0.52	**0.89**	0.32	55	**78**	**7**	116
Meta-Llama-3-8B-Instruct	0.45	0.39	0.57	0.37	63	38	47	108
Random	0.33	0.33	0.33	0.33				

Table 9.4: Performance with **cropped** observations and with **autonomously discovered** subgoals.

Results in Table 9.3 indicate that LLMs can effectively suggest subgoals when presented with game screen observations, and that these subgoals align with those identified by humans. Models like *Meta-Llama-3-70B-Instruct* and *Meta-Llama-3-8B-Instruct* come close to human performance, suggesting that LLMs can effectively use the additional information to suggest and validate subgoals. These results bode well for applications of CALM where human input, while still considerably smaller than in canonical reward shaping, is still expensive to collect.

When transitioning to cropped observations (Table 9.4) LLMs perform worse. This is most likely due to a misalignment between the subgoals proposed by the models and the ones of the ground truth. We did not observe any substantial difference in how different models propose subgoals and in the types of subgoals they suggest. Most models correctly identify *item collection* and *locating objects*, such as stairs, monsters and keys. They often include "going to <object>" instructions as subgoals. We provide examples of such prompts in Appendix A.4.

While this evaluation can be unfair, since we compare the LLM's response with the set of subgoals the human identified, it still tells whether the LLM way of reasoning about a task align with the human one. These elements, together with the ability of LLMs to verify if a subgoal is achieved, suggest that LLMs can be an effective means to transfer human knowledge into value functions.

9.8 Conclusions, limitations, and future work

In this study, we explored whether LLMs can be a useful means to transfer human knowledge into the value function of RL agents. By focusing on reward shaping, we highlighted its limitations in scalability due to the cost of human involvement. To mitigate these costs, we proposed replacing humans with LLMs, leveraging their ability to decompose tasks into shorter subgoals. Preliminary results from an offline dataset of MiniHack demonstrations suggest that LLMs are effective in verifying subgoal achievement and align with those proposed by humans. This suggests the potential of using LLMs to enhance CA in RL.

9.8.1 Limitations of the current evidence

While preliminary results are promising, they are limited by the scope of the current evidence. We did not conduct RL experiments to validate the method in online RL settings. The dynamic nature of online RL could pose unique challenges not present in offline settings. Additionally, despite KeyRoom being representative of the CA challenges, and a common testbed for CA, evaluating the method in a broader range of environments would provide more comprehensive evidence of its robustness and applicability.

9.8.2 Limitations of the method

The method also has inherent limitations. Environments must provide observations in the form of text. The LLM must hold enough knowledge of the game to evaluate actions. While this can be a mild assumption for MiniHack, it can be an obstacle for environments requiring more specialised knowledge, such as Nethack (Küttler et al., 2020) or Crafter (Hafner, 2021; Matthews et al., 2024). Indeed, the LLM relies solely on their prior knowledge and does not incorporate new knowledge while assigning credit, limiting their adaptability and accuracy over time.

Notice that when the prompt does not include the action, as shown in ablation study in Appendix A.6.2, the method is independent of the action set. This means that the method can be applied to environments with a large action space without modifications, because the LLM provides an evaluation of the transition without knowing the action taken in between the two states. However, while scaling to large action spaces is feasible in theory, and the assignment accurate, in practice the influence of each action diminishes as the number of actions increases, as explained in Chapter 5.

We limited the investigation to using only one past state from the history of observations to evaluate the transition. While evidence shows that this is sufficient in practice for the environments considered, it is not clear whether this is enough for more complex environments. However, here we prioritised the simplest settings in which the method can be applied to avoid overstressing the LLM with a context that is too long. In fact, providing a longer transitions history would require a larger context window, with the risk of overwhelming the LLM with too many tokens, and potentially letting the useuful information drown in the noise. To relax the assumption of a single transition, a natural extension of this work is to extend the method to multistep transitions. Here, by including past observations in the prompt beyond the previous one, the LLM can evaluate the current actions (and not *all* the actions in the trajectory). This would align with the current theory practice in Deep RL to use complete histories of observations to evaluate the current action.

One potential pattern of failure of using LLMs as reward shapers is *reward hacking* (Amodei et al., 2016). Reward hacking occurs when the agent learns to exploit the reward function by finding loopholes in the subgoal definitions, rather than achieving the intended goals. In particular, since the LLM is a pretrained model, it is can be vulnerable to adversarial counterexamples (Goodfellow et al., 2014), which can pose serious risks for safety and reliability. For example, if LLM is tricked into providing a very long list of subgoals that are not necessary waypoints to the final goal, the agent might get addicted to collecting rewards from these subgoals, especially if the same subgoal can be achieved multiple times. This can lead to suboptimal policies that do not align with the original task, and result in the agent failing to achieve the intended goal.

Another common pattern of failure of CALM, due to the use of LLMs, is *hallucinations* (Maynez et al., 2020). Hallucination occurs when the LLM generates information that is not present in the input, or that is not grounded in reality. In the context of CALM, for example, the LLM can sometimes fails to correctly understand the grid layout of the environment, or to correctly identify the objects in the observation, most likely due to the shift in the distribution of the training data: observations are not natural language. While we try to investigate potential reasons for this in the ablation study in Appendix A.6.2, A.6.1, it is still an open question how to mitigate this issue.

9.8.3 Future work

Future work should focus on addressing these limitations. Validating the approach in online RL settings and exploring its applicability to a broader range of environments can tell if CALM can enhance the learning process of RL agents in practice.

A natural extension of this work is to generalise the method beyond text-only observations. Baumli et al. (2023) follows this line of research, testing the capability of Vision Language Models (LLMs) to evaluate the completion of an instruction from pixels alone. The instruction completion question corresponds to ours in the LLMs domain.

Finally, a closed feedback loop where CALM helps improve the policy, the policy provides new information to the LLM, and the LLM incorporates this information to improve its CA ability could help scale to more complex problems requiring specialistic knowledge.

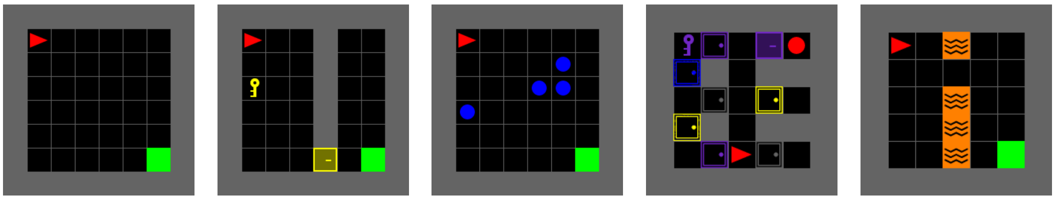
Chapter 10

NAVIX: Scaling MiniGrid with JAX

As Deep RL research moves towards solving large-scale worlds, efficient environment simulations become crucial for rapid experimentation. However, most existing environments struggle to scale. Interactions are typically computed on the CPU, limiting training speed and throughput, due to slower computation and communication overhead when distributing the task across multiple machines. Recently, a set of GPU-based environments has sparked raising interest, proposing a JAX-based, batched re-implementations of common RL environments that significantly increase the throughput of canonical Deep RL algorithms, and enabling large scale parallelism. These environments allow training thousands of agents simultaneously on a single accelerator, vastly outperforming traditional CPU-based environments. In this work, we focus on MiniGrid, a foundational, open-source set of environments that faces the aforementioned limitations. We introduce NAVIX[1], a re-implementation of MiniGrid in JAX. NAVIX achieves over $200\,000\times$ throughput improvements in batch mode, supporting up to 2048 agents in parallel on a single Nvidia A100 80 GB, compared to the multi-threaded implementation of MiniGrid run on

[1]https://github.com/epignatelli/navix

CPU, which can only support up to 16 actors. This reduces experiment times from one week to 15 minutes, promoting faster design iterations and more scalable RL model development.



(a) 128.98× **(b)** 26.47× **(c)** 41.17× **(d)** 19.72× **(e)** 45.71×

Figure 10.1: Speedups for five of the NAVIX environments with respect to their MiniGrid equivalent, using the protocol in Section 10.4.1. (a) `Empty-8x8-v0`, (b) `DoorKey-8x8-v0`, (c) `Dynamic-Obstacles-8x8-v0`, (d) `KeyCorridorS3R3-v0`, (e) `LavaGapS7-v0`.

10.1 Introduction

Deep Reinforcement Learning (Deep RL) is notoriously sample inefficient (Kaiser et al., 2019; Wang et al., 2021a; Johnson et al., 2016; Küttler et al., 2020). Depending on the complexity of the environment dynamics, the observation space, and the action space, agents often require between 10^7 to 10^9 interactions or even more for training. Therefore, as Deep RL moves towards tackling more complex environments, leveraging an efficient environment implementation is an essential ingredient of rapid experimentation and fast design iterations.

However, while the efficiency and scalability of solutions for *agents* have improved massively in recent years (Schulman et al., 2017; Espeholt et al., 2018; Kapturowski et al., 2018), especially due to the scalability of the current deep learning frameworks (Abadi et al., 2016; Paszke et al., 2019; Ansel et al., 2024; Bradbury et al., 2018; Sabne, 2020), environments have not kept pace. They are mostly based on CPU, cannot adapt to different types of devices, and scaling often requires complex distributed systems, introducing design complexity and communication overhead. Overall, deep RL experiments are CPU-bound, limiting both speed and throughput of RL training.

Recently, a set of GPU-based environments (Freeman et al., 2021; Lange, 2022; Weng et al., 2022; Koyamada et al., 2023; Rutherford et al., 2023a; Nikulin et al., 2023; Matthews et al., 2024; Bonnet et al., 2024; Lu et al., 2023; Liesen et al., 2024b) and frameworks (Lu et al., 2022; Liesen et al., 2024a; Toledo, 2024; Nishimori, 2024; Jiang et al., 2023a) has sparked raising interest, proposing JAX-based, batched implementations of common RL environments that can significantly increase the speed and throughput of canonical Deep RL algorithms. This enables large-scale parallelism, allowing the training of thousands of agents in parallel on a single accelerator, significantly outperforming traditional CPU-based environments, and fostering meta-RL applications.

In this work, we build on this trend and focus on the MiniGrid suite of environments (Chevalier-Boisvert et al., 2024), due to its central role in the Deep RL literature. MiniGrid is fundamental to many studies. For instance, Zhang et al. (2020b); Zha et al. (2021); Mavor-Parker et al. (2022) used it to test new exploration strategies; Jiang et al. (2021a) for curriculum learning; Zhao et al. (2021) for planning; Paischer et al. (2022) for representation learning, Flet-Berliac et al. (2021); Guan et al. (2022) for diversity. Parisi et al. (2021) employed MiniGrid to design meta and transfer learning strategies, and Mu et al. (2022) to study language grounding.

However, despite its ubiquity in the Deep RL literature, MiniGrid faces the limitations of CPU-bound environments. We bridge this gap and propose NAVIX, a reimplementation of Minigrid in JAX that leverages JAX's intermediate language representation to migrate the computation to different accelerators, such as GPUs, and TPUs.

Our results show that a single, unbatched NAVIX environment on GPU runs over 10× faster on average than the original, single-threaded Minigrid implementation on CPU (see Section 10.4.1). In common Deep RL settings, a batched NAVIX environment on GPU increases the training throughput of a PPO agent by over $10^6\times$ compared to the multi-threaded CPU implementation of MiniGrid (see Section 10.4.2), turning 1-week experiments into 15 minutes ones. We show the scaling ability of NAVIX by training over 2048 PPO agents in parallel, each using their own subset of environments, all on a single Nvidia A100 80 GB.

The main contributions of this work are the following:

1. A fully JAX-based implementation of 41 environment configurations that reproduces exactly the original Minigrid MDPs and POMDPs.
2. A description of the design philosophy, the design pattern and principles, the organisation, and the components of NAVIX, which, together with the online documentation[2], form an instruction manual to use and extend NAVIX.
3. A set of RL algorithm baselines for all environments in Section 10.4.3.

10.2 Related work

JAX-based environments. The number of JAX-based reimplementations of common environments is in a bullish trend. Freeman et al. (2021) provide a fully differentiable physics engine for robotics, including MJX, a reimplementation of MujoCo (Todorov et al., 2012). Lange (2022) reimplements several gym (Brockman et al., 2016) environments, including classic control, Bsuite (Osband et al., 2020), and MinAtar (Young & Tian, 2019),

Koyamada et al. (2023) reimplement many board games, including backgammon, chess, shogi, and go. Lu et al. (2023) provides JAX implementations of POPGym (Morad et al., 2023), which contains partially-observed RL environments. Matthews et al. (2024) reimplement Crafter (Hafner, 2021). Bonnet

[2]https://epignatelli/navix

et al. (2024) provides JAX implementations of combinatorial problems frequently encountered in industry, including bin packing, capacitated vehicle routing problem, PacMan, Sokoban, Snake, 2048, Sudoku, and many others. Rutherford et al. (2023b) reimplement a set of multi-agent environments, including a MiniGrid-inspired implementation of the Overcooked benchmark.

Yet, none of these works proposes a reimplementation of Minigrid. Weng et al. (2022) is the only one providing a single environment of the suite, *Empty*, but it is only one of the many, most commonly used environments of the suite, and arguably the simplest one.

Batched MiniGrid−like environments. Two works stand out for they aim to partially reimplement MiniGrid. Jiang et al. (2023a) present `AMaze`, a fully batched implementation of a partially observable maze environment, with MiniGrid−like sprites and observations. However, like Weng et al. (2022), the work does not reimplement the full MiniGrid suite. Nikulin et al. (2023) proposes `XLand-MiniGrid`, a suite of grid-world environments for meta RL. Like (Jiang et al., 2023a), `XLand-MiniGrid` reproduces Minigrid−like observations but focuses on designing a set of composable rules that can be used to generate a wide range of environments, rather than reimplementing the original Minigrid suite.

To conclude, MiniGrid is a fundamental tool for Deep RL experiments, at the base of a high number of studies, as we highlighted in Section 10.1. It is easy to use, easy to extend, and provides a wide range of environments of scalable complexity that are easy to inspect for a clearer understanding of an algorithm dynamics, pitfalls, and strengths.

Nevertheless, none of the works above provides a full, batched reimplementation of Minigrid in JAX that mirrors the original suite in terms of environments, observations, state transitions, and rewards. Instead, we propose a full JAX-based reimplementation of the MiniGrid suite that can be used as a

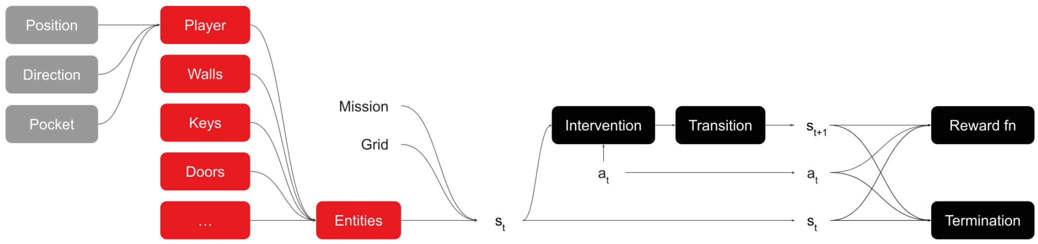


Figure 10.2: Information flow of the ECSM in NAVIX. Entities (*Player, Walls, Keys, Doors, . . .*) are composed of components (*Position, Direction, Pocket*), which hold the data of the entity. Systems (*Intervention, Transition, Rewards, Terminations*) are functions that operate on the collective state of all entities and components.

drop-in replacement for the original environments.

10.3 NAVIX: design philosophy and principles

In this section we describe: ***(i)*** the design philosophy and pattern of NAVIX in Section 10.3.1, and ***(ii)*** the design principles at its foundation in Sections 10.3.2.1 and 10.3.2.2.

In particular, in Section 10.3.2.2, we highlight *why* a JAX port of MiniGrid is not trivial. Among others, the obstacles to transform a stateless program, where a function is allowed to change elements that are not an input of the function, to a stateful one, where the outputs of functions depend solely on the inputs; and the restrictions in the use of `for` loops and control flow primitives, such as `if` statements.[3]

10.3.1 Design pattern

NAVIX is broadly inspired to the ECSM, a design pattern widely used in video game development. In an ECSM, entities – the *objects* on the grid in our case – are composed of components – the *properties* of the object. Each property

[3]See `https://jax.readthedocs.io/en/latest/notebooks/Common_Gotchas_in_JAX.html`.

holds data about the entity, which can then be used to process the game state. For example, an entity `Player` is composed of components `Positionable`, `Holder`, `Directional`, each of which injects properties into the entity: the `Positionable` component injects the `Position` property, holding the coordinates of the entity on the grid, the `Holder` component injects the `Pocket` property, holding the id of the entity that the agent holds, and so on. A full list of components and their properties is provided in Table 10.1. This compositional layout allows to easily generate the wide range of combinations of tasks that MiniGrid offers, and to easily extend the suite with new environments.

Entities are then processed by *systems*, which are functions that operate on the collective state of all entities and components. For example, the *decision* system may update the state of the entities according to the actions taken by a player. The *transition* system may update the state of the entities according to the MDP state transitions. The *observation* system generates the observations that the agents receive, and the *reward* system computes the rewards that the agents receive, and so on. We provide a full list of implemented systems in Appendix A.7.

To develop a better intuition of what these elements are and how they interact, Figure 10.2 shows the information flow of the ECSM in NAVIX.

10.3.2 Design principles

On this background, two principles are at the foundation of NAVIX, and the key aspects that characterise it: ***(i)*** NAVIX aims to exactly match MiniGrid (Section 10.3.2.1), working as a drop-in replacement for the original environments, and; ***(ii)*** every environment is fully jittabile and differentiable (Section 10.3.2.2), to exploit the full set of features that JAX offers.

Component	Property	Shape	Description
`Positionable`	`Position`	`f32[2]`	*Coordinates of the entity on the grid.*
`Directional`	`Direction`	`i32[]`	*Direction of the entity.*
`HasColour`	`Colour`	`u8[]`	*Colour of the entity.*
`Stochastic`	`Probability`	`f32[]`	*Probability that the entity emits an event.*
`Openable`	`State`	`bool[]`	*State of the entity, e.g., open or closed.*
`Pickable`	`Id`	`i32[]`	*Id of the entity that the agent can pick up.*
`HasTag`	`Tag`	`i32[]`	*Categorical value for the entity class.*
`HasSprite`	`Sprite`	`u8[32x32x3]`	*Sprite of the entity in RGB format.*
`Holder`	`Pocket`	`i32[]`	*Id of the entity that the agent holds.*

Table 10.1: List of **Components** in NAVIX. Each component provides a property (or a set of). These properties hold the data that can be accessed and manipulated by the systems (see Table 10.3) to provide observations, rewards, and state transitions.

Entity	Components	Description
`Wall`	`[HasColour]`	*Blocks the agent's movement.*
`Player`	`[Directional, Holder]`	*Can interact with the environment.*
`Goal`	`[HasColour, Stochastic]`	*Can reached to receive a reward.*
`Key`	`[Pickable, HasColour]`	*Can be picked up. Can open doors.*
`Door`	`[Openable, HasColour]`	*Can be opened and closed.*
`Lava`	`[]`	*A room object.*
`Ball`	`[HasColour, Stochastic]`	*A room object.*
`Box`	`[HasColour, Holder]`	*A container of other.*

Table 10.2: List of **Entities** in NAVIX, together with the components that characterise them. By default, all entities already possess `Positionable`, `HasTah`, and `HasSprite` components, in addition to those reported in the table.

System	Function	Description
`Intervention`	$I : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$	*Updates the state using actions.*
`Transition`	$\mu : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$	*Updates the state using MDP dynamics.*
`Observation`	$O : \mathcal{S} \to \mathcal{O}$	*The observation kernel.*
`Reward`	$R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$	*The Markovian reward function.*
`Termination`	$\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{B}$	*The termination function.*

Table 10.3: List of **Systems** in NAVIX. A state $s \in \mathcal{S}$ is a tuple containing: the set of entities, the static grid layout, and the mission of the agent.

10.3.2.1 NAVIX matches MiniGrid

NAVIX matches the original MiniGrid suite in terms of environments, observations, state transitions, rewards, and actions. We include the most commonly used environments of the suite (see Table A.13), and provide a set of baselines for the implemented environments in see Section 10.4 and Table A.13, Appendix A.12.

Formally, a NAVIX environment is a tuple $\mathcal{M} = (h, w, T, \mathcal{O}, \mathcal{A}, \mathcal{R}, d, O, R, \gamma, \mu)$. Here, h and w are the height and width of the grid, T is the number of timesteps before timeout; $\mathcal{O}$ is the observation space, $\mathcal{A}$ is the action space, $\mathcal{R}$ is the reward space; d is the discount factor. O is the observation function, R is the reward function, γ is the termination function, and μ is the transition function.

One key difference between NAVIX and MiniGrid is that the latter uses a non-Markovian reward function. In fact, MiniGrid dispenses a reward of 0 everywhere, except at task completion, where it is inversely proportional to the number of steps taken by the agent to reach the goal:

$$r_t = R(s_t, a, s_{t+1}) - 0.9 * (t + 1)/T, \tag{10.1}$$

Here R is the reward function, s_t is the state at time t, a is the action taken at time t, s_{t+1} is the state at time $t + 1$, and T is the number of timesteps before timeout. Notice the dependency on the number of steps t, which makes the reward non-Markovian.

The use of a non-Markovian reward function is not a mild assumption as most RL algorithms assume Markov rewards. This might call into question the validity of the historical results obtained with MiniGrid, and the generalisation of the results to other environments. For this reason, we depart from the original MiniGrid reward function and use a Markovian reward function, instead, which is 0 everywhere, and 1 at task completion.

10.3.2.2 Stateful and fully jittable

While we aim to match MiniGrid in terms of environments, observations, state transitions, rewards, and actions, the API of NAVIX is different, as it must align with JAX requirements for the environment to be fully jittable. In fact, NAVIX environments can be compiled into XLA and run on any JAX-supported accelerator, including GPUs and TPUs. This includes both simply jitting the `step` function, and jitting the entire training sequence (Lu et al., 2022), assuming that the agent is also implemented in JAX. XLA compilation increases the throughput of experiments massively, allowing for the training of thousands of agents in parallel on a single accelerator, compared to a few that are possible with traditional CPU-based environments. We show the scalability of NAVIX in Section 10.4.

For environments to be fully jittable, the computation must be stateful. For this reason, we need to define an environment *state-object*: the *timestep*. The timestep is a tuple $(t, o_t, a_t, r_{t+1}, \gamma_{t+1}, s_t, i_{t+1})$, where t is the current time – the number of steps elapsed from the last reset – o_t is the observation at time t, a_t is the action taken after o_t, r_{t+1} is the reward received after a_t, γ_{t+1} is the termination signal after a_t, s_t is the true state of the environment at time t, and i_{t+1} is the info dictionary, useful to store accumulations, such as returns.

This structure is necessary to guarantee the same return schema for both the `step` and the `reset` methods, and allows the environment to autoreset, and avoid conditional statements in the agent code, which would prevent the environment from being fully jittable.

At the beginning of the episode, the agent samples a starting state from the starting distribution $\mu_0 : \mathcal{S} \to \mathcal{S}$ using the `reset(key)` method, where `key` is a random key for a stateful random number generator, and the agent receives the first timestep. Since there is no action and reward at the beginning of the episode, we pad with respectively -1, 0. Given an action a_t, the agent can interact with the environment by calling the `step(timestep, action, key)` method. The agent then receives a new state of the environment (a new timestep) and can continue to interact as needed. Code 1 shows an example of how to interact with a jitted NAVIX environment. More examples are provided at `https://epignatelli.com/navix/`.

```python
import navix as nx

# init a NAVIX environment
env = nx.make("Navix-KeyCorridorS6R3-v0")

# sample a starting state
timestep = env.reset(key)
for _ in range(1000):
    # sample a random key
    key, subkey = jax.random.split(key)
    # sample a random action
    action = jax.random.randint(subkey, (1,), 0, 4)
    # interact with the environment
    timestep = jax.jit(env.step)(timestep, action)  # autoresets when done
```

Code 1: Example code to interact with a jitted NAVIX environment.

Notice that the syntax is similar to the original MiniGrid, including the environment *id*, which simply replaces "MiniGrid" with "Navix". The only differences are in the use of an explicit random key for the stateful random number generator, and the fact that the `step` method also takes the current timestep as input, to guarantee the statefulness of the computation.

The schema in Code 1 is an effective template for any kind of agent implementation, including non JAX-jittable agents. However, while this already improves the speed of environment interactions compared to MiniGrid, as shown in Section 10.4.1, the real speed-up comes jitting the whole iteration loop. In Appendix A.8 we provide additional reusable patterns that are useful in daily RL research, such as how to jit the training loop, how to parallelise the training of multiple agents, and how to run hyperparameter search in batch mode.

In addition, in Appendix A.10 we provide a guide on how to extend NAVIX, including new environments, new observations, new rewards, and new termination functions. This is a fundamental aspect to reflect the flexibility of the original MiniGrid suite, which is easy to extend and modify.

10.4 Experiments

This section aims to show the advantages of NAVIX compared to the original MiniGrid implementation, and provides the community with a set of baselines for all environments. It does the former by comparing the two suites, for all environments, both in terms of speed improvements and throughput. For the latter, we train a set of baselines for all environments, and provide a scoreboard that stores the results for all environments. All experiments are run on a single Nvidia A100 80Gb, and Intel(R) Xeon(R) Silver 4310 CPU @ 2.10GHz and 128Gb of RAM.

10.4.1 Speed

We first benchmark the raw speed improvements of NAVIX compared to the original Minigrid implementation, in the most common settings. For each NAVIX environment and its MiniGrid equivalent, we run $1K$ steps with 8 parallel environments, and measure the wall time of both. $1K$ represents the start of the linear regime in Figure 10.4. Notice that this is the mere speed of the environment, and does not include the agent training.

We show results in Figure 10.3 and observe that NAVIX is over $10\times$ faster in unbatched mode than the original MiniGrid implementation on average. These improvements are due to both the migration of the computation to the GPU via XLA, which optimises the computation graph for the specific accelerator, and the batching of the environments. In Figure A.5, Appendix A.12 we ablate the batching, with no parallel environments, and show that the biggest contribution for the speedup is due to efficient batching.

To better understand how the speedup varies with the number of training steps, and to make sure that the $1K$ steps used in the previous experiment are representative of the general trend, we measure the speed improvements for different lenghts of the training runs. We run $1K$, $10K$, $100K$, and $1M$ steps for the `MiniGrid-Empty-8x8-v0` environment and its NAVIX equivalent, and measure the wall time of both.

Results in Figure 10.4 show that NAVIX is consistently faster than the original MiniGrid implementation, regardless of the number of steps. Both MiniGrid show a linear increase in the wall time with the number of steps.

10.4.2 Throughput

While NAVIX provides speed improvements compared to the original MiniGrid implementation, the real advantage comes from the ability to perform highly parallel training runs on a single accelerator. In this experiment, we test how the computation scales with the number of environments.

We first test the limits of NAVIX by measuring the computation while varying the number of environments that run in parallel. MiniGrid uses `gymnasium`, which parallelises the computation with *Python*'s multiprocessing library. NAVIX, instead, uses JAX's native `vmap`, which directly vectorises the computation. We confront the results with the original MiniGrid implementation, using the `MiniGrid-Empty-8x8-v0` environment.

Results in Figure 10.5 show that the original MiniGrid implementation cannot scale beyond 16 environments on 128GB of RAM, for which it takes around 1s to complete $1K$ unrolls. On the contrary, NAVIX can run up to 2^{21} (over $2M$) environments in parallel on the same hardware, with a wall time almost always below 1s. In short, NAVIX achieves a throughput over 10^5 order of magnitude higher than the original MiniGrid implementation.

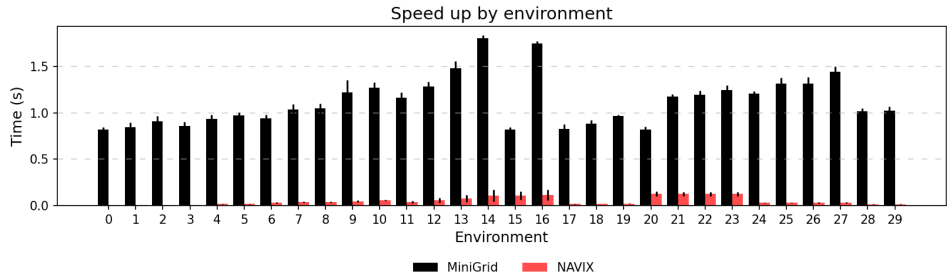


Figure 10.3: Speedup of NAVIX compared to the original Minigrid implementation, for the implemented environments. The identifiers on the x-axis correspond to the environments as reported in Table A.12. Results are the average across 5 runs. Lines show 5-95 percentile confidence intervals. Lower is better.

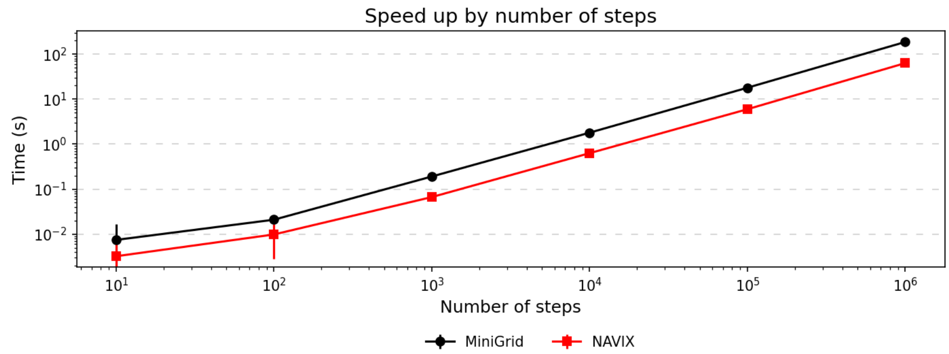


Figure 10.4: Variation of the speedup of NAVIX compared to the original Minigrid implementation according to different numbers of steps for the `MiniGrid-Empty-8x8-v0` environment. Lower is better. Error bars show 5-95 percentile confidence intervals across 5 seeds.

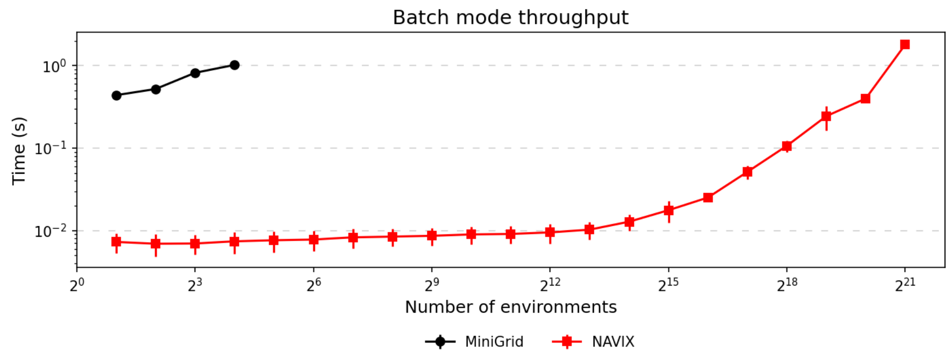


Figure 10.5: Wall time of $1K$ unrolls for both NAVIX and MiniGrid in batch mode.

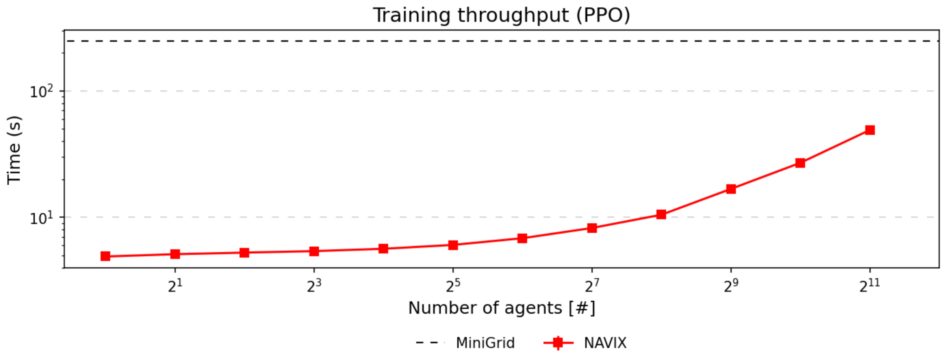


Figure 10.6: Computation costs with growing batch sizes. The agent is a PPO agent on a *Navix-Empty-5x5* environment, run for $1M$ steps across 5 seeds. The effective number of environments is 16 times the number of agents since each PPO agent works on 16 environments.

Secondly, we simulate the very common operation of training many PPO agents, each with their own subset of 16 environments. However, with NAVIX, we do this in parallel. We set the `Empty-8x8-v0` environment, and train the agent for $1M$ steps. Results are shown in Figure 10.6.

We observe that training 2048 agents in NAVIX, for a total of 32 768 environments in parallel, takes less than 50s, almost 5 times faster than the original MiniGrid implementation, which takes around 240s to train a *single* PPO agent. In other words, considering the performance at 2048 agents, NAVIX performs $2048 \times 1M/49s = 668\,734\,693.88$ steps per second ($\sim$ 670 Million steps/s) in batch mode, while the original Minigrid implementation performs $1M/318.01 = 3\,144.65$ steps per second. This is a speedup of over $200\,000\times$.

10.4.3 Baselines

We provide additional baselines using the implementations of PPO (Schulman et al., 2017), Double DQN (DDQN) (Hasselt et al., 2016), and Soft Actor Critic (SAC) (Haarnoja et al., 2018) in Rejax (Liesen et al., 2024a). We optimize hyperparameters (HP) for each algorithm and environment combination using 32 iterations of random search. Each HP configuration is evaluated with 16 different initial seeds. The HP configuration with the highest average final

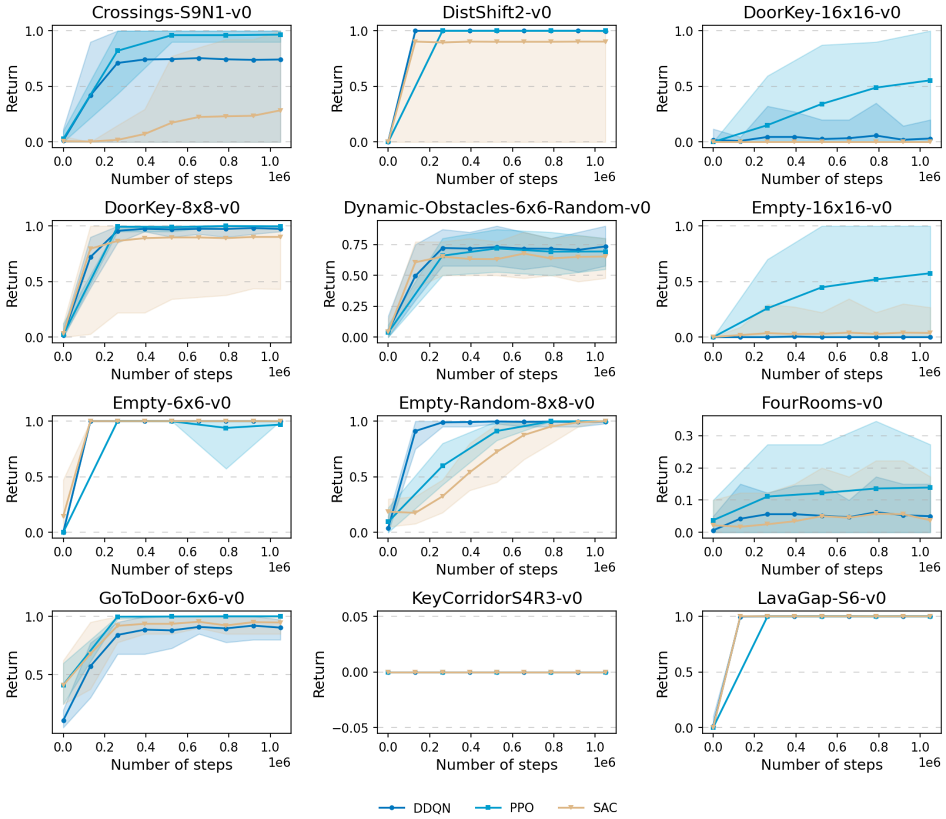


Figure 10.7: Episodic returns for a sample of NAVIX environments for DDQN, PPO and SAC baselines. Lines are average over 32 seeds, and shaded areas show 5-95 percentile confidence intervals.

return is selected. The specific hyperparameters we searched for are shown in Table A.14.

We run the baselines for $10M$ steps, across 32 seeds, with the tuned hyperparameters for the environments shown in Figure 'reffig:baselines. All algorithms use networks with two hidden layers of 64 units. Instead of alternating between a single environment step and network update, DQN and SAC instead perform 128 parallel environment steps and 128 network updates, each with a new minibatch. We found that this significantly improves the runtime while leaving the final performance unaffected.

10.5 Summary, discussion and conclusions

The ability to run fast experiments is crucial to keep up with the state-of-the-art in RL, and to develop new, more efficient algorithms. Recently, the RL community has seen a shift towards more efficient environment implementations, which have historically represented the computational bottleneck of the training process. Among these, JAX has emerged as a powerful tool to accelerate the computation of RL environments. Craftax (Matthews et al., 2024), gymnax (Lange, 2022), JaxMARL Rutherford et al. (2023a), whose relationship with NAVIX we discussed in Section 10.2, are examples of this trend.

In this work, we introduced NAVIX, a reimplementation of the Minigrid environment suite in JAX that leverages JAX's intermediate language representation to migrate the computation to different accelerators, such as GPUs and TPUs. We described the design philosophy, the design pattern, the organisation, and the components of NAVIX, highlighting the connections to the ECSM design pattern, and the correspondence between the structure of its functions and the mathematical formalism of RL.

We presented the environment interface, the list of available environments, and the scoreboard, storing state-of-the-art results that new algorithms can refer to avoid running also baselines, which are prone to errors and manipulations. We showed the speed improvements of NAVIX compared to the original Minigrid implementation, and the scalability of NAVIX with respect to the number of agents that can be trained in parallel, or the number of environments that can be run in parallel. Overall, batched NAVIX environments are over 200 000× faster than the original Minigrid implementation, turning 1-week experiments into 15-minute ones.

These improvements not only enable faster experiments, but they represent a shift of paradigm in the way we think about RL experiments, and the way we design them. For example, large-scale meta-learning experiments, which were previously unfeasible, are now possible. This can unlock very fast searches over a space of algorithms that the designer can explore in a few days, rather than weeks or months. Similarly, because the amount data available to RL agents for a complex environment is mostly limited by the computational budget – and not by the dataset size like in supervised learning – the ability to collect more data in the same amount of time opens the door to large scale experiments that were previously unfeasible.

Part V

Closing

Chapter 11

Conclusions and perspectives

In this thesis, we have investigated the CAP in RL. We have shown that the CAP is a fundamental part of an RL algorithms, and that it plays a key role in improving the performance of the process in delayed and sparse reward settings.

First, we have studied the CAP “problem” in RL from a theoretical perspective. We proposed a new definition of the CAP in RL and showed that it is a fundamental part of the learning process. Then, we have proposed a new algorithm that aims to bridge the gap between the human and the machine learning capabilities in solving POMDPs. Finally, we have re-implemented a seminal RL environment in JAX, which speeds up RL to the point of opening new avenues for research in the field, in addition to a consistent, but bare speedup.

In this chapter, we propose to reflect on this material, discuss it, and advance hypotheses on what the CAP is currently missing to truly scale RL to very complex and real-world problems. This will also constitute a discussion on the limitations of the work presented in this thesis, and the perspectives for future research.

11.1 What are we missing?

We present these reflections in the form of **Calls** that we would like the RL community to consider in future research.

Call 1 (**Current solutions overfit to small problems.**). From all the works referenced in this dissertation, and also from our experiments, we can observe a tendency in the RL literature to focus on small-scale problems. This is understandable, as these problems are easier to solve, and they allow for a more in-depth analysis of the algorithms, leading to a better understanding of the underlying mechanisms. However, none of these works have shown to be able to scale RL to very complex problems: experiments remain limited to small-scale problems, and design choices overfit to these small problems at hand, leading to dead ends. Despite presenting interesting results, this strategy does not seem to lead to improvements that are consistent with the complexity of the problems that we want to solve.

We call for more research whose core contributions do not stop at showing improvements in small-scale problems, but presents clear advancements in solving very complex problems with RL, or at least presents a clear path to scaling RL to these problems.

Call 2 (**The action space is underinvestigated.**). From the material surveyed in **Part III** and from the results presented in Chapter 9, we can observe that there is very little investigation on the impacts of different formulations of the action space on scaling RL to complex problems. Most works focus on how to improve predictions using more accurate, sample efficient or less noisy influence estimators. These very ingenious methods have shown to be very effective in improving the performance of RL algorithms. However, recent advancements in NLP have shown that: *(i)* the design of the action space (the vocabulary in their case) can have a significant impact on the performance of the learning process; and *(ii)* large DNN are very effective when learning large output spaces (for example, discrete distributions with $\geq 50\,000$ events).

There is a need for more research that shows the impacts of large, and potentially more abstract, action spaces on the performance of RL algorithms.

Call 3 (**Options are the way forward.**)**.** One promising area of research for the CAP is the use of options. Currently, there is too much distance between the resolution of the action space in most POMDPs and the level of abstraction at which humans operate. Options are a way to bridge this gap, by providing a way to operate at a higher level of abstraction. However, in the current state of the art, the options that are currently used in RL must be specified *ex-ante* for each problem, which is not scalable to a very large number of problems. Can we learn this options set from data? Maybe from demonstrations? Or can we hard-code a set of options that is general enough to be used in a wide range of problems?

We must find a way to scale options to very complex problems, for example, by learning a set of options from demonstrations, or by hard-coding a set of options that is general enough to be used in a wide range of problems.

Call 4 (**Large-scale foundation models for RL.**)**.** Another lesson that we can learn from the NLP community is the importance of scale. The NLP community has shown that large-scale models can learn very complex patterns, and that they can be used as a foundation for a wide range of tasks. Fine-tuning these models on a specific task has shown to be very effective. How general should this model be, which world should it aim to interact with, and what starting base skills should it have are still open problems.

There is a clear need for a large-scale foundation model for RL, that can be efficiently fine-tuned on a wide range of problems, or that can learn in-context how to adapt to very complex POMDPs.

Call 5 (**We need to leverage demonstrations.**)**.** Demonstrations are a very powerful source of information because they provide a useful, even if sometimes suboptimal, signal on how to solve a problem. This can be very useful in the early stages of learning, when the agent does not know what to explore, improving on both the sample efficiency and the safety of the learning process. It is key that future research focuses on learning from action-less demonstrations, as these are the most abundant form of data that is publicly available.

We call for more research on how to leverage demonstrations to improve the performance of RL algorithms.

Call 6 (**RL is necessary for open-ended learning.**). While demonstrations are a very powerful source of information, they are not enough to face a constant stream of new, unseen problems. The agent must be able to learn from its own experience, and to explore the environment to discover new ways of solving problems. While supervised learning achieving unprecedented results in a wide range of tasks, RL still is plays a fundamental role in scaling AIs to the real world. Aligning to humans, adapting to each human, and learning from new discoveries are a key component of our interaction with the world. If we want to build a safe, useful, and responsible AIs that can interact with the world as we do, we cannot resort to imitation alone, and we must also focus on RL.

Despite its recent bearish trend in publications, RL still has a fundamental role to play in scaling AI to the real world.

We kindly encourage the research community to join in solving these challenges in a shared effort, and we hope that the material collected in this dissertation can be a helpful resource to both inform future advancements in the field, inspire new applications, and ultimately scale RL safely to the real world.

Bibliography

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

David Abel, Cameron Allen, Dilip Arumugam, D Ellis Hershkowitz, Michael L Littman, and Lawson LS Wong. Bad-policy density: A measure of reinforcement learning hardness. *arXiv preprint arXiv:2110.03424*, 2021a.

David Abel, Will Dabney, Anna Harutyunyan, Mark K Ho, Michael Littman, Doina Precup, and Satinder Singh. On the expressivity of markov reward. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021b.

David Abel, Andre Barreto, Benjamin Van Roy, Doina Precup, Hado van Hasselt, and Satinder Singh. A definition of continual reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=ZZS9WEWYbD`.

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: 29304–29320, 2021.

Mostafa Al-Emran. Hierarchical reinforcement learning: a survey. *International journal of computing and digital systems*, 4(02), 2015.

Susan Amin, Maziar Gomrokchi, Harsh Satija, Herke van Hoof, and Doina Precup. A survey of exploration methods in reinforcement learning. *arXiv preprint arXiv:2109.00157*, 2021.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Fredrik Andersen, Rani Lill Anjum, and Elena Rocca. Philosophy of biology: Philosophical bias is the one bias that science cannot avoid. *Elife*, 8:e44929, 2019.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf`.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 929–947, 2024.

Thomas Anthony, Tom Eccles, Andrea Tacchetti, János Kramár, Ian Gemp, Thomas Hudson, Nicolas Porcel, Marc Lanctot, Julien Pérolat, Richard Everett, et al. Learning to play no-press diplomacy with best response policy iteration. *Advances in Neural Information Processing Systems*, 33:17987–18003, 2020.

Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. *Advances in Neural Information Processing Systems*, 32, 2019.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

Kai Arulkumaran, Dylan R Ashley, Jürgen Schmidhuber, and Rupesh K Srivastava. All you need is supervised learning: From imitation learning to meta-rl with upside down rl. *arXiv preprint arXiv:2202.11960*, 2022.

Dilip Arumugam, Peter Henderson, and Pierre-Luc Bacon. An information-theoretic perspective on credit assignment in reinforcement learning. *CoRR*, abs/2103.06224, 2021. URL `https://arxiv.org/abs/2103.06224`.

Dylan R Ashley, Kai Arulkumaran, Jürgen Schmidhuber, and Rupesh Kumar Srivastava. Learning relative return policies with upside-down reinforcement learning. *arXiv preprint arXiv:2202.12742*, 2022.

Karl Johan Åström. Optimal control of markov processes with incomplete state information. *Journal of mathematical analysis and applications*, 10(1): 174–205, 1965.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57:

Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pp. 507–517. PMLR, 2020.

Akhil Bagaria and George Konidaris. Option discovery using deep skill chaining. In *International Conference on Learning Representations*, 2019.

Leemon C III Baird. *Reinforcement Learning Through Gradient Descent*. PhD thesis, US Air Force Academy, 1999.

David Balduzzi, Hastagiri Vanchinathan, and Joachim Buhmann. Kickback cuts backprop's red-tape: Biologically plausible credit assignment in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

Elias Bareinboim, Juan D. Correa, Duligur Ibeling, and Thomas Icard. *On Pearl's Hierarchy and the Foundations of Causal Inference*, pp. 507–556. Association for Computing Machinery, New York, NY, USA, 1 edition, 2022. ISBN 9781450395861. URL `https://doi.org/10.1145/3501714.3501743`.

Andrew G Barto. Adaptive critic and the basal ganglia. In J C Houk, J Davis, and D Beiser (eds.), *Models of Information Processing in the Basal Ganglia*, pp. 215–232. MIT Press, Cambridge, 1995.

Andrew G Barto. Intrinsic motivation and reinforcement learning. *Intrinsically motivated learning in natural and artificial systems*, pp. 17–47, 2013.

Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1):41–77, 2003.

Andrew G Barto, Satinder Singh, Nuttapong Chentanez, et al. Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the 3rd International Conference on Development and Learning*, volume 112, pp. 19. Citeseer, 2004.

Kate Baumli, Satinder Baveja, Feryal Behbahani, Harris Chan, Gheorghe Comanici, Sebastian Flennerhag, Maxime Gazeau, Kristian Holsheimer, Dan

Horgan, Michael Laskin, et al. Vision-language models as a source of rewards. *arXiv preprint arXiv:2312.09187*, 2023.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Vahid Behzadan and William Hsu. Adversarial exploitation of policy imitation. *arXiv preprint arXiv:1906.01121*, 2019.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Marc G Bellemare, Georg Ostrovski, Arthur Guez, Philip Thomas, and Rémi Munos. Increasing the action gap: New operators for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pp. 449–458. Proceedings of Machine Learning Research, 2017.

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, 2020.

R E Bellman. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey, 1957.

Vladimir Grigorevich Boltyansky, Revaz Valerianovic Gamkrelidze, and Lev Semenovich Pontryagin. *On the theory of optimal processes. (In Russian.)*, volume 110. Doklady Akademii Nauk, SSSR, 1956.

Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence I. Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries P. Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn N. Waters, Mohamed A. Mimouni, Ulrich A. Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in jax, 2024. URL `https://arxiv.org/abs/2306.09884`.

Michael Bowling, John D Martin, David Abel, and Will Dabney. Settling the reward hypothesis. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 3003–3020. PMLR, 23–29 Jul 2023. URL `https://proceedings.mlr.press/v202/bowling23a.html`.

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. URL `http://github.com/google/jax`.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. arxiv. *arXiv preprint arXiv:1606.01540*, 10, 2016.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*, pp. 287–318. PMLR, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Bruce G Buchanan. A (very) brief history of artificial intelligence. *Ai Magazine*, 26(4):53–53, 2005.

Lars Buesing, Theophane Weber, Yori Zwols, Nicolas Heess, Sebastien Racaniere, Arthur Guez, and Jean-Baptiste Lespiau. Woulda, coulda, shoulda: Counterfactually-guided policy search. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=BJG0voC9YQ`.

Sean Carroll. Moving naturalism forward. https://www.preposterousuniverse.com/blog/2012/12/11/moving-naturalism-forward-videos-and-recap/, 2012.

Thomas Carta, Pierre-Yves Oudeyer, Olivier Sigaud, and Sylvain Lamprier. Eager: Asking and answering questions for automatic reward shaping in language-guided rl. *Advances in Neural Information Processing Systems*, 35:12478–12490, 2022.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pp. 3676–3713. PMLR, 2023.

Yu-Han Chang, Tracey Ho, and Leslie Kaelbling. All learning is local: Multi-agent learning in global reward games. *Advances in neural information processing systems*, 16, 2003.

Veronica Chelu, Doina Precup, and Hado P van Hasselt. Forethought and hindsight in credit assignment. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 2270–2281. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/18064d61b6f93dab8681a460779b8429-Paper.pdf.

Veronica Chelu, Diana Borsa, Doina Precup, and Hado van Hasselt. Selective credit assignment. *arXiv preprint arXiv:2202.09699*, 2022.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 15084–15097. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/7f489f642a0ddb10272b5c31057f0663-Paper.pdf.

Zhixin Chen and Mengxiang Lin. Self-imitation learning in sparse reward settings. *CoRR, abs/2010.06962*, 2020.

Nuttapong Chentanez, Andrew Barto, and Satinder Singh. Intrinsically motivated reinforcement learning. *Advances in neural information processing systems*, 17, 2004.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Advances in Neural Information Processing Systems*, 36, 2024.

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.

Thomas W Clark. Fear of mechanism. a compatibilist critique of 'the volitional brain'. *Journal of Consciousness Studies*, 6(8-9):279–293, 1999.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International Conference on Machine Learning*, pp. 2048–2056. Proceedings of Machine Learning Research, 2020.

Cédric Colas, Tristan Karch, Olivier Sigaud, and Pierre-Yves Oudeyer. Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: a short survey. *Journal of Artificial Intelligence Research*, 74: 1159–1199, 2022.

Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Murtaza Dalal, Tarun Chiruvolu, Devendra Singh Chaplot, and Ruslan Salakhutdinov. Plan-seq-learn: Language model guided RL for solving long horizon robotics tasks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=hQVCCxQrYN.

Ishita Dasgupta, Christine Kaeser-Chen, Kenneth Marino, Arun Ahuja, Sheila Babayan, Felix Hill, and Rob Fergus. Collaborating with language models for embodied reasoning. *arXiv preprint arXiv:2302.00763*, 2023.

Terrence W Deacon. *Incomplete nature: How mind emerged from matter*. W.W. Norton & Company, New York, New York, 2012.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. arxiv 2023. *arXiv preprint arXiv:2305.14314*, 2023.

Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. In *International Conference on Machine Learning*, pp. 8657–8677. PMLR, 2023.

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. *arXiv preprint arXiv:1512.07679*, 2015.

Ashley D Edwards, Laura Downs, and James C Davidson. Forward-backward reinforcement learning. *arXiv preprint arXiv:1803.10227*, 2018.

Ward Edwards. The theory of decision making. *Psychological bulletin*, 51(4): 380–417, 1954.

Yonathan Efroni, Nadav Merlis, and Shie Mannor. Reinforcement learning with trajectory feedback. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7288–7295, 2021.

Andrew J Elliot and James W Fryer. The goal construct in psychology. In *Handbook of motivation science*, volume 18, pp. 235–250, 2008.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Benjamin Eysenbach and Sergey Levine. Maximum entropy RL (provably) solves some robust RL problems. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=PtSAD3caaA2`.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2018.

Francesco Faccio, Louis Kirsch, and Jürgen Schmidhuber. Parameter-based value functions. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=tV6oBfuyLTQ`.

Amir-massoud Farahmand. Action-gap phenomenon in reinforcement learning. *Advances in Neural Information Processing Systems*, 24, 2011.

Lesley K Fellows. The cognitive neuroscience of human decision making: a review and conceptual framework. *Behavioral and Cognitive Neuroscience Reviews*, 3(3):159–172, 2004.

Johan Ferret. *On Actions that Matter: Credit Assignment and Interpretability in Reinforcement Learning.* PhD thesis, Université de Lille, 2022.

Johan Ferret, Raphaël Marinier, Matthieu Geist, and Olivier Pietquin. Self-attentional credit assignment for transfer in reinforcement learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20, 2021a. ISBN 9780999241165.

Johan Ferret, Olivier Pietquin, and Matthieu Geist. Self-imitation advantage learning. In *AAMAS 2021-20th International Conference on Autonomous Agents and Multiagent Systems*, 2021b.

Angelos Filos, Panagiotis Tigkas, Rowan McAllister, Nicholas Rhinehart, Sergey Levine, and Yarin Gal. Can autonomous vehicles identify, recover from, and adapt to distribution shifts? In *International Conference on Machine Learning*, pp. 3145–3153. PMLR, 2020.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pp. 1126–1135. Proceedings of Machine Learning Research, 2017.

Yannis Flet-Berliac. The promise of hierarchical reinforcement learning. *The Gradient*, 9, 2019.

Yannis Flet-Berliac, Johan Ferret, Olivier Pietquin, Philippe Preux, and Matthieu Geist. Adversarially guided actor-critic. *arXiv preprint arXiv:2102.04376*, 2021.

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.

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. URL `http://github.com/google/brax`.

Hiroki Furuta, Yutaka Matsuo, and Shixiang Shane Gu. Generalized decision transformer for offline hindsight information matching. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=CAjxVodl_v`.

Jim Gao. Machine learning applications for data center optimization. 2014.

Javier García, Fern, and o Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(42):1437–1480, 2015. URL `http://jmlr.org/papers/v16/garcia15a.html`.

Matthieu Geist, Bruno Scherrer, et al. Off-policy learning with eligibility traces: a survey. *J. Mach. Learn. Res.*, 15(1):289–333, 2014.

Paul W Glimcher. Understanding dopamine and reinforcement learning: The dopamine reward prediction error hypothesis. *Proceedings of the National Academy of Sciences*, 108(Supplement 3):15647–15654, 2011. doi: 10.1073/pnas.1014269108. URL `https://www.pnas.org/doi/abs/10.1073/pnas.1014269108`.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Anirudh Goyal, Philemon Brakel, William Fedus, Soumye Singhal, Timothy Lillicrap, Sergey Levine, Hugo Larochelle, and Yoshua Bengio. Recall traces: Backtracking models for efficient reinforcement learning. In *International Conference on Learning Representations*, 2019a. URL `https://openreview.net/forum?id=HygsfnR9Ym`.

Prasoon Goyal, Scott Niekum, and Raymond J Mooney. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*, 2019b.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. *Advances in neural information processing systems*, 28, 2015.

Nathan Grinsztajn, Johan Ferret, Olivier Pietquin, Matthieu Geist, et al. There is no turning back: A self-supervised approach for reversibility-aware reinforcement learning. *Advances in Neural Information Processing Systems*, 34:1898–1911, 2021.

Lin Guan, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging approximate symbolic models for reinforcement learning via skill diversity. In *International Conference on Machine Learning*, pp. 7949–7967. PMLR, 2022.

Arthur Guez, Fabio Viola, Theophane Weber, Lars Buesing, Steven Kapturowski, Doina Precup, David Silver, and Nicolas Heess. Value-driven hindsight modelling. *Advances in Neural Information Processing Systems*, 33: 12499–12509, 2020.

Abhishek Gupta, Aldo Pacchiano, Yuexiang Zhai, Sham Kakade, and Sergey Levine. Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity. *Advances in Neural Information Processing Systems*, 35:15281–15295, 2022.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pp. 1352–1361. PMLR, 2017.

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*, pp. 1861–1870. Proceedings of Machine Learning Research, 2018.

Michael Haenlein and Andreas Kaplan. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California management review*, 61(4):5–14, 2019.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

Danijar Hafner, Kuang-Huei Lee, Ian Fischer, and Pieter Abbeel. Deep hierarchical planning from pixels. *Advances in Neural Information Processing Systems*, 35:26091–26104, 2022.

Jean Harb, Tom Schaul, Doina Precup, and Pierre-Luc Bacon. Policy evaluation networks. *arXiv preprint arXiv:2002.11833*, 2020.

Anna Harutyunyan, Peter Vrancx, Pierre-Luc Bacon, Doina Precup, and Ann Nowe. Learning with options that terminate off-policy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Gheshlaghi Azar, Bilal Piot, Nicolas Heess, Hado P van Hasselt, Gregory Wayne, Satinder Singh, Doina Precup, et al. Hindsight credit assignment. *Advances in neural information processing systems*, 32, 2019.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. AAAI'16, pp. 2094–2100. AAAI Press, 2016.

Donald O Hebb. *The organization of behavior; A neuropsychological theory.* John Wiley, 1949.

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.

Donald D Hoffman. The interface theory of perception. *Current Directions in Psychological Science*, 25(3):157–161, 2016.

James C Houk and James L Adams. A model of how the basal ganglia generate and use neural signals that predict reinforcement. In J C Houk, J Davis, and D Beiser (eds.), *Models of Information Processing in the Basal Ganglia*, pp. 249–270. MIT Press, Cambridge, 1995.

Ronald A Howard. *Dynamic programming and Markov processes.* John Wiley, 1960.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pp. 9118–9147. PMLR, 2022.

Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, et al. Grounded decoding: Guiding text generation with grounded models for robot control. *arXiv preprint arXiv:2303.00855*, 2023.

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, 2019. ISSN 2041-1723. doi: 10.1038/s41467-019-13073-w. URL `https://doi.org/10.1038/s41467-019-13073-w`.

Tommi Jaakkola, Michael Jordan, and Satinder Singh. Convergence of stochastic iterative dynamic programming algorithms. *Advances in neural information processing systems*, 6, 1993.

Athul Paul Jacob, Mike Lewis, and Jacob Andreas. Multitasking inhibits semantic drift. *arXiv preprint arXiv:2104.07219*, 2021.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=SJ6yPD5xg`.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

Dominik Janzing, David Balduzzi, Moritz Grosse-Wentrup, and Bernhard

Schölkopf. Quantifying causal influences. *The Annals Of Statistics*, pp. 2324–2358, 2013.

Stratton C Jaquette. Markov decision processes with a new optimality criterion: Discrete time. *The Annals of Statistics*, 1(3):496–505, 1973.

Khurram Javed and Martha White. Meta-learning representations for continual learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Dominik Jeurissen, Diego Perez-Liebana, Jeremy Gow, Duygu Cakmak, and James Kwan. Playing nethack with llms: Potential & limitations as zero-shot agents. *arXiv preprint arXiv:2403.00690*, 2024.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pp. 4940–4950. Proceedings of Machine Learning Research, 2021a.

Minqi Jiang, Michael Dennis, Edward Grefenstette, and Tim Rocktäschel. minimax: Efficient baselines for autocurricula in jax. 2023a.

Minqi Jiang, Tim Rocktäschel, and Edward Grefenstette. General intelligence requires rethinking exploration. *Royal Society Open Science*, 10(6):230539, 2023b.

Ray Jiang, Tom Zahavy, Zhongwen Xu, Adam White, Matteo Hessel, Charles Blundell, and Hado Van Hasselt. Emphatic algorithms for deep reinforcement learning. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5023–5033. PMLR, 18–24 Jul 2021b. URL `https://proceedings.mlr.press/v139/jiang21j.html`.

Yiding Jiang, Shixiang Shane Gu, Kevin P Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Zhijing Jin, Yuen Chen, Felix Leeb, Luigi Gresele, Ojasv Kamal, Zhiheng Lyu, Kevin Blin, Fernando Gonzalez, Max Kleiman-Weiner, Mrinmaya Sachan, and Bernhard Schölkopf. CLadder: Assessing causal reasoning in language models. In *NeurIPS*, 2023. URL `https://openreview.net/forum?id=e2wtjx0Yqu`.

Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *Ijcai*, volume 16, pp. 4246–4247, 2016.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1xCPJHtDB`.

L. J. Kamin. Predictability, surprise, attention, and conditioning. In B. A. Campbell and R. M. Church (eds.), *Punishment Aversive Behavior*, pp. 279–296. Appleton- Century-Crofts, New York, 1969.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2019.

Steven Kapturowski, Víctor Campos, Ray Jiang, Nemanja Rakicevic, Hado van Hasselt, Charles Blundell, and Adria Puigdomenech Badia. Human-level atari 200x faster. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=JtC6yOHRoJJ`.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE conference on computational intelligence and games (CIG)*, pp. 1–8. IEEE, 2016.

Emre Kıcıman, Robert Ness, Amit Sharma, and Chenhao Tan. Causal reasoning and large language models: Opening a new frontier for causality. *arXiv preprint arXiv:2305.00050*, 2023.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023.

Martin Klissarov and Doina Precup. Flexible option learning. *Advances in Neural Information Processing Systems*, 34:4632–4646, 2021.

Martin Klissarov, Rasool Fakoor, Jonas Mueller, Kavosh Asadi, Taesup Kim, and Alex Smola. Adaptive interest for emphatic reinforcement learning. In *Decision Awareness in Reinforcement Learning Workshop at ICML 2022*, 2022. URL `https://openreview.net/forum?id=ZGi3bDRXkx`.

Martin Klissarov, Pierluca D'Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence feedback. 2023.

A Harry Klopf. Brain function and adaptive systems: a heterostatic theory. Technical Report 133, Air Force Cambridge Research Laboratories. Special Reports, Bedford, Massachusets, 1972.

A. Harry Klopf. A comparison of natural and artificial intelligence. *Special Interest Group on Artificial Intelligence Bulletin*, pp. 11–13, jun 1975.

A Harry Klopf. *The hedonistic neuron: A theory of memory, learning, and intelligence*. Taylor & Francis Inc, 1982.

A Harry Klopf. A neuronal model of classical conditioning. *Psychobiology*, 16 (2):85–125, 1988.

Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3): 122–148, 2013.

Sotetsu Koyamada, Shinri Okano, Soichiro Nishimori, Yu Murata, Keigo Habara, Haruka Kita, and Shin Ishii. Pgx: Hardware-accelerated parallel game simulators for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 45716–45743, 2023.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models, 2023.

Robert Tjarko Lange. gymnax: A JAX-based reinforcement learning environment library. 2022. URL `http://github.com/RobertTLange/gymnax`.

Kennon A Lattal. Delayed reinforcement of operant behavior. *Journal of the Experimental Analysis of Behavior*, 93(1):129–139, 2010.

Charline Le Lan, Stephen Tu, Adam Oberman, Rishabh Agarwal, and Marc G Bellemare. On the generalization of representations in reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 4132–4157. PMLR, 2022.

Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, Lisa Lee, Daniel Freeman, Winnie Xu, Sergio Guadarrama, Ian Fischer, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *arXiv preprint arXiv:2205.15241*, 2022.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control, 2022. *URL https://arxiv. org/abs/2209.07753*, 3, 2022.

Jarek Liesen, Chris Lu, and Robert Lange. rejax, 2024a. URL `https://github.com/keraJLi/rejax`.

Jarek Liesen, Chris Lu, Andrei Lupu, Jakob N Foerster, Henning Sprekeler, and Robert T Lange. Discovering minimal reinforcement learning environments. *arXiv preprint arXiv:2406.12589*, 2024b.

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.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321, 1992.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. In *International Conference on Learning Representations*, 2017.

Minghuan Liu, Menghui Zhu, and Weinan Zhang. Goal-conditioned reinforcement learning: Problems and solutions. *arXiv preprint arXiv:2201.08299*, 2022.

Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35:16455–16468, 2022.

Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Foerster, Satinder Singh, and Feryal Behbahani. Structured state space models for in-context reinforcement learning. *arXiv preprint arXiv:2303.03982*, 2023.

J Luketina, N Nardelli, G Farquhar, J Foerster, J Andreas, E Grefenstette, S Whiteson, and T Rocktäschel. A survey of reinforcement learning informed by natural language. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, August 10-16 2019, Macao, China.*, volume 57, pp. 6309–6317. AAAI Press (Association for the Advancement of Artificial Intelligence), 2019.

Jukka Luoma, Sampsa Ruutu, Adelaide Wilcox King, and Henrikki Tikkanen. Time delays, competitive interdependence, and firm performance. *Strategic Management Journal*, 38(3):506–525, 2017.

A Rupam Mahmood, Huizhen Yu, Martha White, and Richard S Sutton. Emphatic temporal-difference learning. *arXiv preprint arXiv:1507.01569*, 2015.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. *arXiv preprint arXiv:2402.16801*, 2024.

Augustine Mavor-Parker, Kimberly Young, Caswell Barry, and Lewis Griffin. How to stay curious while avoiding noisy tvs using aleatoric uncertainty estimation. In *International Conference on Machine Learning*, pp. 15220–15240. PMLR, 2022.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization. *arXiv preprint arXiv:2005.00661*, 2020.

Luigi F Menabrea and AA Lovelace. Sketch of the analytical engine invented by charles babbage, esq., by lf menabrea, of turin, officer of the military engineers. *Translated and with notes by AA L. Taylor's Scientific Memoirs*, 3:666–731, 1843.

Matheus RF Mendonca, Artur Ziviani, and André MS Barreto. Graph-based skill acquisition for reinforcement learning. *ACM Computing Surveys (CSUR)*, 52(1):1–26, 2019.

Thomas Mesnard, Theophane Weber, Fabio Viola, Shantanu Thakoor, Alaa Saade, Anna Harutyunyan, Will Dabney, Thomas S Stepleton, Nicolas Heess, Arthur Guez, et al. Counterfactual credit assignment in model-free reinforcement learning. In *International Conference on Machine Learning*, pp. 7654–7664. Proceedings of Machine Learning Research, 2021.

Thomas Mesnard, Wenqi Chen, Alaa Saade, Yunhao Tang, Mark Rowland, Theophane Weber, Clare Lyle, Audrunas Gruslys, Michal Valko, Will Dabney, Georg Ostrovski, Eric Moulines, and Remi Munos. Quantile credit assignment. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 24517–24531. PMLR, 23–29 Jul 2023. URL `https://proceedings.mlr.press/v202/mesnard23a.html`.

Donald Michie. Experiments on the mechanization of game-learning part i. characterization of the model and its parameters. *The Computer Journal*, 6 (3):232–236, 1963.

Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961. ISSN 00968390. doi: 10.1109/JRPROC.1961.287775.

Marvin Minsky. *Semantic information processing*. The MIT Press, Cambridge, Massachusetts, 1968.

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.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, Deep Learning Workshop, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

P Read Montague, Peter Dayan, and Terrence J Sejnowski. A framework for mesencephalic dopamine systems based on predictive hebbian learning. *Journal of neuroscience*, 16(5):1936–1947, 1996.

Steven Morad, Ryan Kortvelesy, Matteo Bettini, Stephan Liwicki, and Amanda Prorok. POPGym: Benchmarking partially observable reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=chDrutUTs0K`.

Seyed Sajad Mousavi, Michael Schukat, Enda Howley, and Patrick Mannion. Applying q (λ)-learning in deep reinforcement learning to play atari games. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, pp. 1–6, 2017.

Jesse Mu, Victor Zhong, Roberta Raileanu, Minqi Jiang, Noah Goodman, Tim Rocktäschel, and Edward Grefenstette. Improving intrinsic exploration with language abstractions. *Advances in Neural Information Processing Systems*, 35:33947–33960, 2022.

Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *Advances in neural information processing systems*, 31, 2018.

Suraj Nair, Mohammad Babaeizadeh, Chelsea Finn, Sergey Levine, and Vikash Kumar. Trass: Time reversal as self-supervision. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 115–121. IEEE, 2020.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287. Citeseer, 1999.

Thanh Thi Nguyen and Vijay Janapa Reddi. Deep reinforcement learning for cyber security. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Tianwei Ni, Michel Ma, Benjamin Eysenbach, and Pierre-Luc Bacon. When do transformers shine in RL? decoupling memory from credit assignment. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=APGXBNkt6h`.

Alexander Nikulin, Vladislav Kurenkov, Ilya Zisman, Viacheslav Sinii, Artem Agarkov, and Sergey Kolesnikov. XLand-minigrid: Scalable meta-reinforcement learning environments in JAX. In *Intrinsically-Motivated and Open-Ended Learning Workshop, NeurIPS2023*, 2023. URL `https://openreview.net/forum?id=xALDC4aHGz`.

Soichiro Nishimori. Jax-corl: Clean sigle-file implementations of offline rl algorithms in jax. 2024. URL `https://github.com/nissymori/JAX-CORL`.

Chris Nota, Philip Thomas, and Bruno C. Da Silva. Posterior value functions: Hindsight baselines for policy gradient methods. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8238–8247. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/nota21a.html`.

John O'Doherty, Peter Dayan, Johannes Schultz, Ralf Deichmann, Karl Friston, and Raymond J Dolan. Dissociable roles of ventral and dorsal striatum in instrumental conditioning. *Science*, 304(5669):452–454, 2004.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *International Conference on Machine Learning*, pp. 3878–3887. PMLR, 2018.

Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvári, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado van Hasselt. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rygf-kSYwH`.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Fabian Paischer, Thomas Adler, Vihang Patil, Angela Bitto-Nemling, Markus Holzleitner, Sebastian Lehner, Hamid Eghbal-Zadeh, and Sepp Hochreiter. History compression via language models in reinforcement learning. In *International Conference on Machine Learning*, pp. 17156–17185. PMLR, 2022.

Fabian Paischer, Thomas Adler, Markus Hofmarcher, and Sepp Hochreiter.

Semantic helm: A human-readable memory for reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Hsiao-Ru Pan, Nico Gürtler, Alexander Neitz, and Bernhard Schölkopf. Direct advantage estimation. *Advances in Neural Information Processing Systems*, 35:11869–11880, 2022.

Simone Parisi, Victoria Dean, Deepak Pathak, and Abhinav Gupta. Interesting object, curious agent: Learning task-agnostic exploration. *Advances in Neural Information Processing Systems*, 34:20516–20530, 2021.

Jack Parker-Holder, Aldo Pacchiano, Krzysztof M Choromanski, and Stephen J Roberts. Effective diversity in population based reinforcement learning. *Advances in Neural Information Processing Systems*, 33:18050–18062, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

P. I. Pavlov. *Conditioned Reflexes*. Oxford University Press, London, UK, 1927.

Judea Pearl. Causal inference in statistics: An overview. *Statistics Surveys*, 3: 96–146, 2009.

Judea Pearl et al. Models, reasoning and inference. *Cambridge, UK: CambridgeUniversityPress*, 19(2):3, 2000.

Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas

Anthony, et al. Mastering the game of stratego with model-free multiagent reinforcement learning. *Science*, 378(6623):990–996, 2022.

Jean Piaget. *Play, Dreams and Imitation*. Norton & Company, New York, New York, 1962.

Jean Piaget, Margaret Cook, et al. *The origins of intelligence in children*, volume 8. International Universities Press New York, 1952.

Eduardo Pignatelli, Johan Ferret, Matthieu Geist, Thomas Mesnard, Hado van Hasselt, and Laura Toni. A survey of temporal credit assignment in deep reinforcement learning. *Transactions on Machine Learning Research*, 2024a. ISSN 2835-8856. URL `https://openreview.net/forum?id=bNtr6SLgZf`. Survey Certification.

Eduardo Pignatelli, Johan Ferret, Davide Paglieri, Samuel Coward, Tim Rocktäschel, Edward Grefenstette, and Laura Toni. Assessing the zero-shot capabilities of LLMs for action evaluation in RL. In *ICML Workshop on Automated Reinforcement Learning*, 2024b. URL `https://openreview.net/forum?id=MFw8K5705I`.

Eduardo Pignatelli, Jarek Liesen, Robert Tjarko Lange, Chris Lu, Pablo Samuel Castro, and Laura Toni. Navix: Scaling minigrid environments with jax. *arXiv preprint arXiv:2407.19396*, 2024c. In submission to ICML 2025.

Silviu Pitis. Rethinking the discount factor in reinforcement learning: A decision theoretic approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 7949–7956, 2019.

Silviu Pitis, Elliot Creager, and Animesh Garg. Counterfactual data augmentation using locally factored dynamics. *Advances in Neural Information Processing Systems*, 33:3976–3990, 2020.

Semenovich Pontryagin, Lev, Grigorevich Boltayanskii, Vladimir, Valerianovic Gamkrelidze, Revaz, and Frolovich Mishchenko, Evgeniĭ. *The mathematical theory of optimal processes.* Whiley, 1962.

Chetan Prakash, Kyle D Stephens, Donald D Hoffman, Manish Singh, and Chris Fields. Fitness beats truth in the evolution of perception. *Acta Biotheoretica*, 69:319–341, 2021.

Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, pp. 80, 2000a.

Doina Precup. *Temporal abstraction in reinforcement learning.* PhD thesis, University of Massachusetts Amherst, 2000b.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 2014.

Martin L Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24 (11):1127–1137, 1978.

S R Quartz, Peter Dayan, P R Montague, and TJ Sejnowski. Expectation learning in the brain using diffuse ascending projections. *Society for Neuroscience*, 18(1210), 1993.

Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Hazhir Rahmandad, Nelson Repenning, and John Sterman. Effects of feedback delay on learning. *System Dynamics Review*, 25(4):309–338, 2009.

David Raposo, Sam Ritter, Adam Santoro, Greg Wayne, Theophane Weber, Matt Botvinick, Hado van Hasselt, and Francis Song. Synthetic returns for long-term credit assignment. *CoRR*, 2021.

Paulo Rauber, Avinash Ummadisingu, Filipe Mutz, and Jürgen Schmidhuber. Hindsight policy gradients. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Bkg2viA5FQ`.

R. A. Recorla and A. R. Wagner. A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In A. H. Black and W. F. Prokasy (eds.), *Classical Conditioning II: Current Research and Theory*, pp. 64–99. Appleton- Century-Crofts, New York, 1972.

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.

Zhizhou Ren, Ruihan Guo, Yuan Zhou, and Jian Peng. Learning long-term reward redistribution via randomized return decomposition. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=lpkGn3k2YdD`.

Matthew Riemer, Miao Liu, and Gerald Tesauro. Learning abstract options. *Advances in neural information processing systems*, 31, 2018.

Mark Rowland, Will Dabney, and Rémi Munos. Adaptive trade-offs in off-policy learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 34–44. PMLR, 2020.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardar Ingvarsson, Timon Willi, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, Robert Tjarko Lange, Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktaschel, Chris Lu, and Jakob Nicolaus Foerster. Jaxmarl: Multi-agent rl environments in jax. *arXiv preprint arXiv:2311.10090*, 2023a.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardar Ingvarsson, Timon Willi, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, et al. Jaxmarl: Multi-agent rl environments in jax. *arXiv preprint arXiv:2311.10090*, 2023b.

Amit Sabne. Xla : Compiling machine learning for peak performance, 2020.

Arthur L Samuel. Some studies in machine learning using the game of checkers. ii—recent progress. *IBM Journal of research and development*, 11(6):601–617, 1967.

Arthur Lee Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959. doi: 10.1147/rd.33.0210.

Paul A Samuelson. A note on measurement of utility. *The review of economic studies*, 4(2):155–161, 1937.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Kuttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL `https://openreview.net/forum?id=skFwlyefkWJ`.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320, Lille, France, 07–09 Jul 2015a. PMLR. URL `https://proceedings.mlr.press/v37/schaul15.html`.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015b.

Tom Schaul, André Barreto, John Quan, and Georg Ostrovski. The phenomenon of policy churn. *arXiv preprint arXiv:2206.00730*, 2022.

Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, and Matthieu Geist. Approximate modified policy iteration and its application to the game of tetris. *Journal of Machine Learning Research*, 16(49):1629–1676, 2015.

Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897. Proceedings of Machine Learning Research, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Donald G Schultz. State functions and linear control systems. *McGraw-Hill Book Company*, 1967.

Wolfram Schultz. Predictive reward signal of dopamine neurons. *Journal of neurophysiology*, 80(1):1–27, 1998.

Wolfram Schultz, Peter Dayan, and P Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

Minah Seo, Luiz Felipe Vecchietti, Sangkeum Lee, and Dongsoo Har. Rewards prediction-based credit assignment for reinforcement learning with sparse binary rewards. *IEEE Access*, 7:118776–118791, 2019.

Dhruv Shah, Błażej Osiński, Sergey Levine, et al. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *Conference on robot learning*, pp. 492–504. PMLR, 2023.

Mehran Shakerinava and Siamak Ravanbakhsh. Utility theory for sequential decision making. In *International Conference on Machine Learning*, pp. 19616–19625. PMLR, 2022.

Claude E Shannon. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41 (314):256–275, 1950.

Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. Skill induction and planning with latent language. *arXiv preprint arXiv:2110.01517*, 2021.

Baba Shiv, Antoine Bechara, Irwin Levin, Joseph W Alba, James R Bettman, Laurette Dube, Alice Isen, Barbara Mellers, Ale Smidts, Susan J Grant, et al. Decision neuroscience. *Marketing Letters*, 16(3):375–386, 2005.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419): 1140–1144, 2018.

David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, 2021.

Herbert A Simon. *Models of man; social and rational.* Wiley, New York, 1957.

Herbert A Simon. *Theories of decision-making in economics and behavioural science*, volume 49. Springer, 1959a.

Herbert A Simon. Theories of decision-making in economics and behavioural science. *The American Economic Review*, 49(3):253–283, 1959b.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530. IEEE, 2023.

Satinder Singh, Richard L Lewis, and Andrew G Barto. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pp. 2601–2606. Cognitive Science Society, 2009.

Satinder P Singh and Richard S Sutton. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1):123–158, 1996.

Burrhus F Skinner. Operant behavior. *American psychologist*, 18(8):503, 1963.

Burrhus Frederic Skinner. *Science and Human Behavior*. Macmillan, New York, 1st edition, 1953.

Jane Smiley. The man who invented the computer: The biography of john atanasoff. *Digital Pioneer*, pp. 431–87, 2010.

Matthew Smith, Herke Hoof, and Joelle Pineau. An inference-based policy gradient method for learning options. In *International Conference on Machine Learning*, pp. 4703–4712. PMLR, 2018.

Matthew J Sobel. The variance of discounted markov decision processes. *Journal of Applied Probability*, 19(4):794–802, 1982.

Jonathan Daniel Sorg. *The optimal reward problem: Designing effective reward for bounded agents*. PhD thesis, University of Michigan, 2011.

Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaskowski, and Jürgen Schmidhuber. Training agents using upside-down reinforcement learning. *CoRR*, abs/1912.02877, 2019. URL `http://arxiv.org/abs/1912.02877`.

Miroslav Štrupl, Francesco Faccio, Dylan R Ashley, Jürgen Schmidhuber, and Rupesh Kumar Srivastava. Upside-down reinforcement learning can diverge in stochastic environments with episodic resets. *arXiv preprint arXiv:2205.06595*, 2022.

Ke Sun, Bei Jiang, and Linglong Kong. How does value distribution in distributional reinforcement learning help optimization? *arXiv preprint arXiv:2209.14513*, 2022.

Richard S Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts, 1984.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

Richard S. Sutton. The reward hypothesis. `http://incompleteideas.net/rlai.cs.ualberta.ca/RLAI/rewardhypothesis.html`, 2004.

Richard S Sutton and Andrew G Barto. Toward a modern theory of adaptive networks: expectation and prediction. *Psychological review*, 88(2):135, 1981.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: an introduction*. MIT Press, 2nd edition, 2018. ISBN 9781626239777. URL `http://incompleteideas.net/book/RLbook2020.pdf`.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 761–768, 2011.

Richard S Sutton, A Rupam Mahmood, and Martha White. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, 17(1):2603–2631, 2016.

Yuji Takahashi, Geoffrey Schoenbaum, and Yael Niv. Silencing the critics: understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an actor/critic model. *Frontiers in neuroscience*, pp. 14, 2008.

Yunhao Tang and Alp Kucukelbir. Hindsight expectation maximization for goal-conditioned reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 2863–2871. PMLR, 2021.

Edward Lee Thorndike. Animal intelligence: An experimental study of the associative processes in animals. *American Journal of Psychology*, 2(4), 1898.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

Edan Toledo. Stoix: Distributed Single-Agent Reinforcement Learning End-to-End in JAX. April 2024. URL `https://github.com/EdanToledo/Stoix`.

A M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

Hado van Hasselt and Richard S Sutton. Learning to predict independent of span. *arXiv preprint arXiv:1508.04582*, 2015.

Hado van Hasselt and Marco A Wiering. Using continuous action spaces to solve discrete problems. In *2009 International Joint Conference on Neural Networks*, pp. 1149–1156. IEEE, 2009.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

Hado van Hasselt, Sephora Madjiheurem, Matteo Hessel, David Silver, André Barreto, and Diana Borsa. Expected eligibility traces. In *Proceedings of* *the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9997–10005, 2021.

Hado P van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32, 2019.

Oshin Vartanian and David R Mandel. *Neuroscience of Decision Making*. Psychology Press, New York, 2011.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

David Venuto, Elaine Lau, Doina Precup, and Ofir Nachum. Policy gradients incorporating the future. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=EHaUTlm2eHg`.

Nino Vieillard, Tadashi Kozuno, Bruno Scherrer, Olivier Pietquin, Rémi Munos, and Matthieu Geist. Leverage the average: an analysis of kl regularization in reinforcement learning. *Advances in Neural Information Processing Systems*, 33:12163–12174, 2020a.

Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Munchausen reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 4235–4246, 2020b.

Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray

Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

Jane X Wang, Michael King, Nicolas Pierre Mickael Porcel, Zeb Kurth-Nelson, Tina Zhu, Charlie Deck, Peter Choy, Mary Cassin, Malcolm Reynolds, H. Francis Song, Gavin Buttimore, David P Reichert, Neil Charles Rabinowitz, Loic Matthey, Demis Hassabis, Alexander Lerchner, and Matthew Botvinick. Alchemy: A benchmark and analysis toolkit for meta-reinforcement learning agents. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021a. URL `https://openreview.net/forum?id=eZu4BZxlRnX`.

Jianhao Wang, Wenzhe Li, Haozhe Jiang, Guangxiang Zhu, Siyuan Li, and Chongjie Zhang. Offline reinforcement learning with reverse model-based imagination. *Advances in Neural Information Processing Systems*, 34:29420–29432, 2021b.

Tony Tong Wang, Adam Gleave, Nora Belrose, Tom Tseng, Joseph Miller, Michael D Dennis, Yawen Duan, Viktor Pogrebniak, Sergey Levine, and Stuart Russell. Adversarial policies beat professional-level go ais. *arXiv preprint arXiv:2211.00241*, 2022.

Zhe Wang and Tianzhen Hong. Reinforcement learning for building controls: The opportunities and challenges. *Applied Energy*, 269:115036, 2020.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. In *International Conference on Learning Representations*, 2016a.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016b.

Christopher J C H Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, United Kingdom, 1989.

Christopher J C H Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, et al. Envpool: A highly parallel reinforcement learning environment execution engine. *Advances in Neural Information Processing Systems*, 35:22409–22421, 2022.

Douglas J White. Mean, variance, and probabilistic criteria in finite markov decision processes: A review. *Journal of Optimization Theory and Applications*, 56:1–29, 1988.

Martha White. Unifying task specification in reinforcement learning. In *International Conference on Machine Learning*, pp. 3742–3750. PMLR, 2017.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Yue Wu, Yewen Fan, Paul Pu Liang, Amos Azaria, Yuanzhi Li, and Tom M Mitchell. Read and reap the rewards: Learning to play atari with the help of instruction manuals. *Advances in Neural Information Processing Systems*, 36, 2024.

Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.

Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

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.

Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. Keep calm and explore: Language models for action generation in text-based games. *arXiv preprint arXiv:2010.02903*, 2020.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34:25476–25488, 2021.

Haiyan Yin, Shuicheng YAN, and Zhongwen Xu. Distributional meta-gradient reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=LGkmUauBUL`.

Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

Daochen Zha, Wenye Ma, Lei Yuan, Xia Hu, and Ji Liu. Rank the episodes: A simple approach for exploration in procedurally-generated environments. *arXiv preprint arXiv:2101.08152*, 2021.

Shangtong Zhang, Vivek Veeriah, and Shimon Whiteson. Learning retrospective knowledge with reverse reinforcement learning. *Advances in Neural Information Processing Systems*, 33:19976–19987, 2020a.

Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian. Bebold: Exploration beyond the boundary of explored regions. *arXiv preprint arXiv:2012.08621*, 2020b.

Mingde Zhao, Zhen Liu, Sitao Luan, Shuyuan Zhang, Doina Precup, and Yoshua Bengio. A consciousness-inspired planning agent for model-based reinforcement learning. *Advances in neural information processing systems*, 34:1569–1581, 2021.

Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 27042–27059. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/zheng22c.html`.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. *Advances in Neural Information Processing Systems*, 31, 2018.

Zeyu Zheng, Junhyuk Oh, Matteo Hessel, Zhongwen Xu, Manuel Kroiss, Hado van Hasselt, David Silver, and Satinder Singh. What can learned intrinsic rewards capture? In *International Conference on Machine Learning*, pp. 11436–11446. PMLR, 2020.

Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. Rtfm: Generalising to new environment dynamics via reading. In *ICLR*, pp. 1–17. ICLR, 2020.

Victor Zhong, Jesse Mu, Luke Zettlemoyer, Edward Grefenstette, and Tim Rocktäschel. Improving policy learning via language dynamics distillation. *Advances in Neural Information Processing Systems*, 35:12504–12515, 2022.

Appendix A

A.1 Further related works

The literature also offers surveys on related topics. Liu et al. (2022) review challenges and solutions of Goal-Conditioned Reinforcement Learning (GCRL), and Colas et al. (2022) follow to extend GCRL to Intrinsically Motivated Goal Exploration Process (IMGEP). Both these works are relevant for they generalise RL to multiple goals, but while goal-conditioning is a key ingredient of further arguments (see Section 4.2), GCRL does not aim to address CAP directly. Barto & Mahadevan (2003); Al-Emran (2015); Mendonca et al. (2019); Flet-Berliac (2019); Pateria et al. (2021) survey HRL. HRL breaks down a long-term task into a hierarchical set of smaller sub-tasks, where each sub-task can be interpreted as an independent goal. However, despite sub-tasks providing intermediate, mid-way feedback that reduces the overall delay of effects that characterises the CAP, these works on HRL are limited to investigate the CAP only by decomposing the problem into smaller ones. Even in these cases, for example in the case of temporally abstract actions (Sutton et al., 1999), sub-tasks either are not always well defined, or they require strong domain knowledge that might hinder generalisation.

A.2 Further details on contexts

A *contextual distribution* defines a general mechanism to collect the contextual data c (experience). For example, it can be a set of predefined demonstration, an MDP to actively query by interaction, or imaginary rollouts produced by an internal world model. This is a key ingredient of each method, together with its choice of action influence and the protocol to learn that from experience. Two algorithms can use the same action influence measure (e.g., (Klopf, 1972) and (Goyal et al., 2019a)), but specify different contextual distributions, resulting in two separate, often very different methods.

Formally, we represent a context as a distribution over some contextual data $C \sim \mathbb{P}_C(C)$, where C is the context, and $\mathbb{P}_C$ is the distribution induced by a specific choice of source. Our main reference for the classification of contextual distributions is the *ladder of causality* (Pearl, 2009; Bareinboim et al., 2022), seeing, doing, imagining, and we define our three classes accordingly.

Observational distributions are distributions over a predefined set of data, and we denote it with $\mathbb{P}_{obs}(C)$. Here, the agent has only access to passive set of experience collection from a (possibly unknown) environment. It cannot intervene or affect the environment in any way, but it must learn from the data that is available: it cannot explore. This is the typical case of offline CA methods or methods that learn from demonstrations (Chen et al., 2021), where the context is a fixed dataset of trajectories. The agent can sample from $\mathbb{P}_{obs}$ uniformly at random or with forms of prioritisation (Schaul et al., 2015b; Jiang et al., 2021a). Observational distributions allow assigning credit efficiently and safely since they do not require direct interactions with the environment and can ignore the burden of either waiting for the environment to respond or getting stuck into irreversible states (Grinsztajn et al., 2021). However, they can be limited both in the amount of information they can provide and in the overall coverage of the space of associations between actions and outcomes, often failing to generalise to unobserved associations (Kirk et al., 2023).

Interactive distributions are distributions defined by active interactions with an environment, and we denote them with $\mathbb{P}_{\mu,\pi}$. Here, the agent can actively intervene to control the environment through the policy, which defines a distribution over trajectories, $D \sim \mathbb{P}_{\mu,\pi}$. This is the typical case of model-free, online CA methods (Arjona-Medina et al., 2019; Harutyunyan et al., 2019), where the source is the interface of interaction between the agent and the environment. Interactive distributions allow the agent to make informed decisions about which experience to collect (Amin et al., 2021) because the space of associations between actions and outcomes is under the direct control of the agent: they allow exploration. One interesting use of these distributions is to define outcomes in *hindsight*, that is, by unrolling the policy in the environment with a prior objective and then considering a different goal from the resulting trajectory (Andrychowicz et al., 2017). Interactive distributions provide greater information than observational ones but may be more expensive to query, they do not allow to specify all queries, such as starting from a specific state or crossing the MDP backwards, and they might lead to irreversible outcomes with safety concerns (García et al., 2015).

Hypothetical distributions are distributions defined by functions internal to the agent, and we denote them with $\mathbb{P}_{\widetilde{\mu},\pi}$, where $\widetilde{\mu}$ is the agent's internal state-transition dynamic function (learned). They represent potential scenarios, futures or pasts, that do not correspond to actual data collected from the real environment. The agent can query the space of associations surgically and explore a broader space of possible outcomes for a given action without having to interact with the environment. In short, it can imagine a hypothetical scenario, and reason about what would have happened if the agent had taken a different action. Hypothetical distributions enable counterfactual reasoning, that is, to reason about what would have happened if the agent had taken a different action in a given situation. Crucially, they allow navigating the

MDP independently of the arrow of time, and, for example, pause the process of generating a trajectory, revert to a previous state, and then continue the trajectory from that point. However, they can produce a paradoxical situation in which the agent explores a region of space with high uncertainty, but relies on a world model that, because of that uncertainty is not very accurate (Guez et al., 2020).

A.2.1 Representing a context

Since equation (4.1) includes a context as an input a natural question arises, "*How to represent contexts?*". Recall that the purpose of the context is to two-fold: *a)*to unambiguously determine the current present as much as possible, and *b)* to convey information about the distribution of actions that will be taken *after* the action we aim to evaluate. Section 4.3 details the reasons of the choice. In many action influence measures (see Section 4.6), such as q-values or advantage, the context is only the state of an MDP, or a history if we are solving a POMDP instead. In this case representing the context is the problem of representing a state, which is widely discussed in literature. Notice that this is not about *learning* a state representation, but rather about specifying the shape of a state when constructing and defining an MDP or an observation and an action for a POMDP. These portion of the input addresses the first purpose of a context.

To fulfil its second function the context may contain additional objects and here we discuss only the documented cases, rather than proposing a theoretical generalisation. When the additional input is a policy (Harb et al., 2020; Faccio et al., 2021), then the problem turns to how to represent that specific object. In the specific case of a policy Harb et al. (2020) and Faccio et al. (2021) propose two different methods of representing a policy. In other cases, future actions are specified using a full trajectory, or a feature of it, and in this case the evaluation happens in hindsight. As for policies, the problem turns to representing this additional portion of the context.

A.3 Prompting

To develop an intuition of the LLM's task, we show examples of prompts for each configuration used in the experiments of Section 9.7. In particular, we show examples of prompts for both option termination verification and option discovery, and both game screen and cropped observations. Finally, we present prompts where two options terminated (a subgoal is achieved): a key pick-up and a door unlock.

A.3.1 Cropped vs gamescreen observations

Prompt with cropped observations

```
The environment is MiniHack.

I will present you with a short extract of a gameplay. At each timestep, symbols represent the following items:
- "." represents a floor tile.
- "|" can represent either a wall, a vertical wall, an open door.
- "-" can represent either the bottom left corner (of a room), bottom right corner (of a room), wall, horizontal wall,
wall, top left corner (of a room), op right corner (of a room).
- "+" represents a closed door. Doors can be locked, and require a key to open.
- "(" represents a useful item (pick-axe, key, lamp...)
- "<" represents a ladder or staircase up.
- ">" represents a ladder or staircase down.

The task of the agent is to win the game.

First, based on your knowledge of NetHack, break down the task of the agent into subgoals.
Then, consider the following game transition, which might or might not contain these subgoals.
Determine if any of the subgoals is achieved at Time: 1 or not.


Report your response in a dictionary containing the name of the subgoals as keys and booleans as value. For example:
```python
{
    <name of goal>: <bool>,
}

Observation Sequence:

<gameplay>
Time: 0
Current message: Never mind.


      - - - -
      | . . |
      | . . |
      - @ < |
  . . . . . |
| . ( . . . |
- - - - - - -

Time: 1
Current message:



      - - - -
      | . . |
      | @ . |
      - . < |
  . . . . . |
| . ( . . . |
- - - - - - -

</gameplay>

I will not consider anything that is not in the dictionary.
You have only one shot at this, and you cannot ask for clarifications.
```

Prompt 2: Example of a prompt where transitions have cropped observations.

Prompt with gamescreen observations

```
The environment is MiniHack.

I will present you with a short extract of a gameplay. At each timestep, symbols represent the following items:
- "." represents a floor tile.
- "|" can represent either a wall, a vertical wall, an open door.
- "-" can represent either the bottom left corner (of a room), bottom right corner (of a room), wall, horizontal wall,
wall, top left corner (of a room), op right corner (of a room).
- "+" represents a closed door. Doors can be locked, and require a key to open.
- "(" represents a useful item (pick-axe, key, lamp...)
- "<" represents a ladder or staircase up.
- ">" represents a ladder or staircase down.

The task of the agent is to win the game.

First, based on your knowledge of NetHack, break down the task of the agent into subgoals.
Then, consider the following game transition, which might or might not contain these subgoals.
Determine if any of the subgoals is achieved at Time: 1 or not.


Report your response in a dictionary containing the name of the subgoals as keys and booleans as value. For example:
```python
{
    <name of goal>: <bool>,
}

Observation Sequence:

<gameplay>
Time: 0

Never mind.
```

A.3.2 Option termination vs option discovery

Prompt with cropped observations

```
The environment is MiniHack.

I will present you with a short extract of a gameplay. At each timestep, symbols represent the following items:
- "." represents a floor tile.
- "|" can represent either a wall, a vertical wall, an open door.
- "-" can represent either the bottom left corner (of a room), bottom right corner (of a room), wall, horizontal wall,
wall, top left corner (of a room), op right corner (of a room).
- "+" represents a closed door. Doors can be locked, and require a key to open.
- "(" represents a useful item (pick-axe, key, lamp...)
- "<" represents a ladder or staircase up.
- ">" represents a ladder or staircase down.

The task of the agent is to win the game.

Consider the following subgoals:
‘‘‘python
subgoals = {
    "pick up the key": None,
    "open the door": None,
}
‘‘‘
Then, consider the following game transition, which might or might not contain these subgoals.
Determine if any of the subgoals is achieved at Time: 1 or not.


Report your response in a dictionary containing the name of the subgoals as keys and booleans as value. For example:
‘‘‘python
{
    <name of goal>: <bool>,
}

Observation Sequence:

<gameplay>
Time: 0
Current message: Never mind.


      - - - -
      | . . |
      | . . |
      - @ < |
  . . . . . |
| . ( . . . |
- - - - - - -

Time: 1
Current message:



      - - - -
      | . . |
      | @ . |
      - . < |
  . . . . . |
| . ( . . . |
- - - - - - -

</gameplay>

I will not consider anything that is not in the dictionary.
You have only one shot at this, and you cannot ask for clarifications.
```

Prompt 4: Example of a prompt for instruction verification. Here, goals are provided externally from a human.

Prompt with cropped observations

```
The environment is MiniHack.

I will present you with a short extract of a gameplay. At each timestep, symbols represent the following items:
- "." represents a floor tile.
- "|" can represent either a wall, a vertical wall, an open door.
- "-" can represent either the bottom left corner (of a room), bottom right corner (of a room), wall, horizontal wall,
wall, top left corner (of a room), op right corner (of a room).
- "+" represents a closed door. Doors can be locked, and require a key to open.
- "(" represents a useful item (pick-axe, key, lamp...)
- "<" represents a ladder or staircase up.
- ">" represents a ladder or staircase down.

The task of the agent is to win the game.

First, based on your knowledge of NetHack, break down the task of the agent into subgoals.
Then, consider the following game transition, which might or might not contain these subgoals.
Determine if any of the subgoals is achieved at Time: 1 or not.


Report your response in a dictionary containing the name of the subgoals as keys and booleans as value. For example:
```python
{
    <name of goal>: <bool>,
}

Observation Sequence:

<gameplay>
Time: 0
Current message: Never mind.


      - - - -
      | . . |
      | . . |
      - @ < |
  . . . . . |
| . ( . . . |
- - - - - - -

Time: 1
Current message:



      - - - -
      | . . |
      | @ . |
      - . < |
  . . . . . |
| . ( . . . |
- - - - - - -

</gameplay>

I will not consider anything that is not in the dictionary.
You have only one shot at this, and you cannot ask for clarifications.
```

Prompt 5: Example of a prompt for options discovery and termination verification.

A.3.3 Examples of different subgoals
```

Examples of key pickup

```
The environment is MiniHack.

I will present you with a short extract of a gameplay. At each timestep, symbols represent the following items:
- "." represents a floor tile.
- "|" can represent either a wall, a vertical wall, an open door.
- "-" can represent either the bottom left corner (of a room), bottom right corner (of a room), wall, horizontal wall,
wall, top left corner (of a room), op right corner (of a room).
- "+" represents a closed door. Doors can be locked, and require a key to open.
- "(" represents a useful item (pick-axe, key, lamp...)
- "<" represents a ladder or staircase up.
- ">" represents a ladder or staircase down.

The task of the agent is to win the game.

First, based on your knowledge of NetHack, break down the task of the agent into subgoals.
Then, consider the following game transition, which might or might not contain these subgoals.
Determine if any of the subgoals is achieved at Time: 1 or not.


Report your response in a dictionary containing the name of the subgoals as keys and booleans as value. For example:
```python
{
    <name of goal>: <bool>,
}

Observation Sequence:

<gameplay>
Time: 0
Current message: It's a wall.

          | . . |
          | . . |
    - - + - . < |
    | . . . . . |
    | . @ . . . |
    - - - - - - -



Time: 1
Current message: g - a key named The Master Key of Thievery.

          | . . |
          | . . |
    - - + - . < |
    | . . . . . |
    | . @ . . . |
    - - - - - - -




</gameplay>

I will not consider anything that is not in the dictionary.
You have only one shot at this, and you cannot ask for clarifications.
```

Prompt 6: Example of a prompt where the transition shows a successful key pickup subgoal.
```

Examples of door unlock

```
The environment is MiniHack.

I will present you with a short extract of a gameplay. At each timestep, symbols represent the following items:
- "." represents a floor tile.
- "|" can represent either a wall, a vertical wall, an open door.
- "-" can represent either the bottom left corner (of a room), bottom right corner (of a room), wall, horizontal wall,
wall, top left corner (of a room), op right corner (of a room).
- "+" represents a closed door. Doors can be locked, and require a key to open.
- "(" represents a useful item (pick-axe, key, lamp...)
- "<" represents a ladder or staircase up.
- ">" represents a ladder or staircase down.

The task of the agent is to win the game.

First, based on your knowledge of NetHack, break down the task of the agent into subgoals.
Then, consider the following game transition, which might or might not contain these subgoals.
Determine if any of the subgoals is achieved at Time: 1 or not.


Report your response in a dictionary containing the name of the subgoals as keys and booleans as value. For example:
```python
{
    <name of goal>: <bool>,
}

Observation Sequence:

<gameplay>
Time: 0
Current message:

        - - - -
        | . . |
        | . . |
  - - + - . < |
  | . . @ . . |
  | . . . . . |
  - - - - - - -


Time: 1
Current message: Never mind.

  - -   - - - -
  | >   | . . |
    . . | . . |
  - - | - . < |
  | . . @ . . |
  | . . . . . |
  - - - - - - -



</gameplay>

I will not consider anything that is not in the dictionary.
You have only one shot at this, and you cannot ask for clarifications.
```

Prompt 7: Example of a prompt where the transition shows a successful door-unlock subgoal.
```

A.4 Responses

To give an intuition of how different models approach the task, here, we show a series of responses to Prompt 1. Notice that, to stimulate the model to reason about subgoals, we ask to consider its knowledge of Nethack, rather than MiniHack.

c4ai-command-r-v01

```
```python
{
    'find_item': False,
    'open_door': False,
    'collect_item': False,
    'move_upstairs': False,
    'move_downstairs': True,
    'kill_monster': False,
    'find_key': False,
    'open_lock': False,
    'light_lamp': False,
    'win_game': False
}
```
```

Response 1: Example response

gemma-1.1-7b-it

```
## Subgoals:

- **Find a way out:** Locate and reach the exit of the map.
- **Gather useful items:** Collect any available items for future use.
- **Explore the environment:** Discover the layout of the map and potential hazards.


## Subgoal Achieved at Time: 1

```python
{
    "Find a way out": False,
    "Gather useful items": True,
    "Explore the environment": True,
}
```
```

Response 2: Example response

Mistral-7B-Instruct-v0.2

```
Based on my understanding of the given environment and the task, the subgoals for the agent are as follows:

1. Exploration: Reach all the unvisited parts of the map.
2. Collection: Gather all the useful items (pick-axe, key, lamp, etc.)
3. Door Opening: Open all the doors (locked or not) to progress through the map.

Now, let's analyze the given gameplay sequence and determine if any of the subgoals have been achieved at Time: 1.

```python
{
    "Exploration": False,
    "Collection": False,
    "Door Opening": False
}
```

At Time: 1, the agent has moved to a new location and has not interacted with any items or doors. Therefore, none of
the subgoals have been achieved at this point.
```

Response 3: Example response

Mixtral-8x7B-Instruct-v0.1

```
Based on my knowledge of NetHack, the task of the agent can be broken down into the following subgoals:

1. Movement: The agent needs to move through the game environment to reach the goal.
2. Collection: The agent needs to collect useful items that can help it in achieving the goal.
3. Door manipulation: The agent needs to open doors to explore new areas.
4. Staircase navigation: The agent needs to navigate staircases to move to different levels of the game.
5. Key collection: The agent needs to collect keys to open locked doors.
6. Winning the game: The agent needs to reach the final goal and win the game.

Based on the given game transition, the subgoals achieved at Time: 1 are:

‘‘‘python
{
    "Movement": True,
    "Collection": False,
    "Door manipulation": False,
    "Staircase navigation": False,
    "Key collection": False,
    "Winning the game": False,
}
‘‘‘

Explanation:

* Movement: The agent moved from the bottom left corner of the room to the bottom right corner of the room, so this
subgoal is achieved.
* Collection: The agent did not collect any item in this transition, so this subgoal is not achieved.
* Door manipulation: The agent did not encounter any doors in this transition, so this subgoal is not achieved.
* Staircase navigation: The agent did not encounter any staircases in this transition, so this subgoal is not
achieved.
* Key collection: The agent did not collect any keys in this transition, so this subgoal is not achieved.
* Winning the game: The agent did not reach the final goal in this transition, so this subgoal is not achieved.
```

Response 4: Example response

Llama-3-8B-Instruct

```
Based on my knowledge of NetHack, I break down the task of the agent into the following subgoals:

* Find a useful item (pick-axe, key, lamp...)
* Find a ladder or staircase up
* Find a ladder or staircase down
* Open a door (if it’s closed)
* Unlock a door (if it’s locked)
* Reach the goal (win the game)

Now, considering the given game transition, I determine if any of the subgoals has been achieved at Time: 1 or not:

{
    ’Find a useful item’: False,
    ’Find a ladder or staircase up’: False,
    ’Find a ladder or staircase down’: True,
    ’Open a door (if it\’s closed)’: False,
    ’Unlock a door (if it\’s locked)’: False,
    ’Reach the goal’: False
}

Note that the agent has moved north and then east, but no new items or doors have been discovered, and no doors have
been opened or unlocked. However, the agent is now closer to the ladder or staircase down, which is a step towards
reaching the goal.
```

Response 5: Example response

Llama-3-70B-Instruct

```
Based on my knowledge of NetHack, I break down the task of the agent into the following subgoals:

* Reach the staircase/ladder: The agent needs to find a way to move up or down the levels to progress in the game.
* Collect useful items: The agent needs to collect items such as pick-axes, keys, and lamps to overcome obstacles and
access new areas.
* Unlock doors: The agent needs to find keys to unlock closed doors and access new areas.
* Avoid obstacles: The agent needs to avoid walls, locked doors, and other obstacles that can hinder its progress.
* Reach the goal: The agent needs to reach the final goal, which is not specified in this extract but is typically the
treasure or the exit.

Now, considering the given game transition, I determine if any of the subgoals have been achieved at Time: 1 or not:

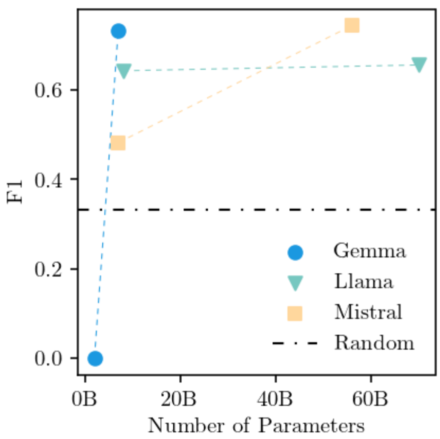
{
    'Reach the staircase/ladder': False,
    'Collect useful items': False,
    'Unlock doors': False,
    'Avoid obstacles': True,
    'Reach the goal': False,
}

The agent has avoided obstacles by moving north and not hitting the wall, but none of the other subgoals have been
achieved at Time: 1.
```

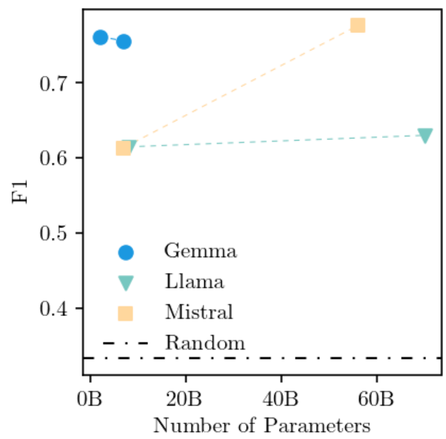
Response 6: Example prompt for option discovery, where goals are not provided externally.

A.5 Scale

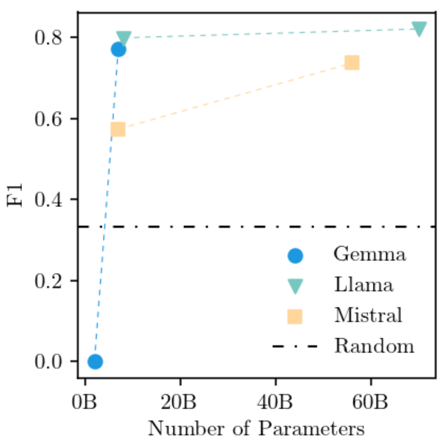
In this section we show how the F1 score varies according to the number of parameters of the LLM. For a fair analysis, we only compare models of the same family, where multiple sizes of the model are available. Namely, we consider: *Meta-Llama-3-8B-Instruct* and *Meta-Llama-3-70B-Instruct*; *gemma-1.1-2b-it* and *gemma-1.1-7b-it*; *Mistral-7B-Instruct-v0.2* and *Mixtral-8x7B-Instruct-v0.1*. Since *c4ai-command-r-plus* does not fit an Nvidia A100 80Gb, we exclude *c4ai-command-r-v01* from the comparison.



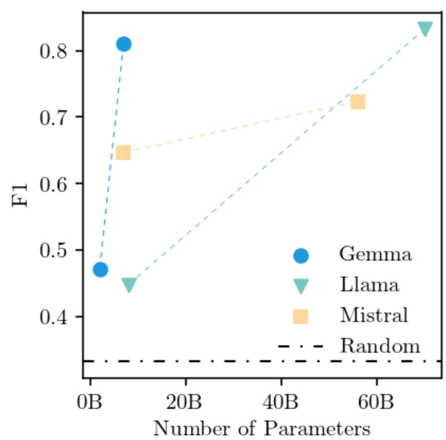
(a)



(b)



(c)



(d)

Figure A.1: F1 score as a function of the LLM size.

A.6 Ablations

In this section, we include results of ablations, to understand which part of the prompt affect the performance the most. We present two ablations:

(i) With and without a token separator to isolate each cell in the grid observation.

(ii) With and without including actions in the transition.

A.6.1 Tokenisation

Annotator	F1 ↑	Accuracy ↑	Precision ↑	Recall ↑	TP ↑	TN ↑	FP ↓	FN ↓
Human	**1.00**	**1.00**	**1.00**	**1.00**	**171**	**85**	**0**	**0**
Mixtral-8x7B-Instruct-v0.1	**0.73**	**0.65**	0.75	**0.71**	**121**	45	40	**50**
c4ai-command-r-v01	0.66	0.63	0.87	0.53	90	72	13	81
gemma-1.1-7b-it	0.66	0.64	0.89	0.52	89	74	11	82
Mistral-7B-Instruct-v0.2	0.58	0.60	0.97	0.41	70	83	2	101
Meta-Llama-3-8B-Instruct	0.54	0.56	0.90	0.39	66	78	7	105
gemma-1.1-2b-it	0.53	0.49	0.69	0.43	74	52	33	97
Meta-Llama-3-70B-Instruct	0.16	0.39	**1.00**	0.09	15	**85**	**0**	156
c4ai-command-r-plus	0.02	0.33	0.40	0.01	2	82	3	169
Random	0.33	0.33	0.33	0.33				

Table A.1: Performance of LLM annotations with **game screen** observations, subgoals **provided** in the prompt, and **no token separator**.

Annotator	F1 ↑	Accuracy ↑	Precision ↑	Recall ↑	TP ↑	TN ↑	FP ↓	FN ↓
Human	**1.00**	**1.00**	**1.00**	**1.00**	**171**	**85**	**0**	**0**
Mixtral-8x7B-Instruct-v0.1	**0.76**	0.68	0.77	**0.75**	**128**	47	38	**43**
gemma-1.1-7b-it	**0.76**	0.68	0.77	**0.75**	**128**	46	39	**43**
c4ai-command-r-v01	0.74	**0.69**	0.83	0.67	115	62	23	56
Meta-Llama-3-70B-Instruct	0.71	0.68	0.89	0.60	102	72	13	69
gemma-1.1-2b-it	0.68	0.57	0.67	0.68	116	29	56	55
c4ai-command-r-plus	0.64	0.61	0.84	0.51	88	68	17	83
Mistral-7B-Instruct-v0.2	0.59	0.61	**0.99**	0.42	72	**84**	**1**	99
Meta-Llama-3-8B-Instruct	0.49	0.52	0.87	0.34	58	76	9	113
Random	0.33	0.33	0.33	0.33				

Table A.2: Performance of LLM annotations with **cropped** observations, subgoals **provided** in the prompt, and **no token separator**.

Examples of prompt with no token separator

```
The environment is MiniHack.

I will present you with a short extract of a gameplay. At each timestep, symbols represent the following items:
- "." represents a floor tile.
- "|" can represent either a wall, a vertical wall, an open door.
- "-" can represent either the bottom left corner (of a room), bottom right corner (of a room), wall, horizontal wall,
wall, top left corner (of a room), op right corner (of a room).
- "+" represents a closed door. Doors can be locked, and require a key to open.
- "(" represents a useful item (pick-axe, key, lamp...)
- "<" represents a ladder or staircase up.
- ">" represents a ladder or staircase down.

The task of the agent is to win the game.

First, based on your knowledge of NetHack, break down the task of the agent into subgoals.
Then, consider the following game transition, which might or might not contain these subgoals.
Determine if any of the subgoals is achieved at Time: 1 or not.


Report your response in a dictionary containing the name of the subgoals as keys and booleans as value. For example:
```python
{
    <name of goal>: <bool>,
}

Observation Sequence:

<gameplay>
Time: 0
Current message: Never mind.


   ----
   |..|
   |..|
   -@<|
 .....|
|.(...|
-------

Time: 1
Current message:



   ----
   |..|
   |@.|
   -.<|
 .....|
|.(...|
-------

</gameplay>

I will not consider anything that is not in the dictionary.
You have only one shot at this, and you cannot ask for clarifications.
```

Prompt 8: Example of a prompt where the transition shows a successful door-unlock subgoal.

Annotator	F1 ↑	Accuracy ↑	Precision ↑	Recall ↑	TP ↑	TN ↑	FP ↓	FN ↓
Human	**1.00**	**1.00**	**1.00**	**1.00**	**171**	**85**	**0**	**0**
Meta-Llama-3-70B-Instruct	**0.81**	0.71	0.72	**0.92**	**157**	25	60	**14**
c4ai-command-r-plus	0.79	**0.74**	**0.86**	0.73	125	64	21	46
gemma-1.1-7b-it	0.75	0.63	0.68	0.84	144	17	68	27
Mixtral-8x7B-Instruct-v0.1	0.72	0.61	0.69	0.76	130	27	58	41
c4ai-command-r-v01	0.63	0.58	0.76	0.54	92	56	29	79
Meta-Llama-3-8B-Instruct	0.56	0.56	**0.86**	0.41	70	74	11	101
Mistral-7B-Instruct-v0.2	0.52	0.54	0.83	0.38	65	72	13	106
gemma-1.1-2b-it	0.00	0.33	0.00	0.00	0	**85**	**0**	171
Random	0.33	0.33	0.33	0.33				

Table A.3: Performance of LLM annotations with **game screen** observations, subgoals **suggested** by the LLM, and **no token separator**.
```

Annotator	F1 ↑	Accuracy ↑	Precision ↑	Recall ↑	TP ↑	TN ↑	FP ↓	FN ↓
Human	**1.00**	**1.00**	**1.00**	**1.00**	**171**	**85**	**0**	**0**
Meta-Llama-3-70B-Instruct	**0.83**	**0.74**	0.74	0.94	161	29	56	10
gemma-1.1-7b-it	**0.83**	0.72	0.71	**0.98**	**168**	17	68	**3**
Mixtral-8x7B-Instruct-v0.1	0.78	0.68	0.72	0.85	146	27	58	25
c4ai-command-r-plus	0.75	0.69	**0.81**	0.71	121	56	29	50
Meta-Llama-3-8B-Instruct	0.69	0.59	0.70	0.68	116	35	50	55
c4ai-command-r-v01	0.66	0.57	0.70	0.63	107	39	46	64
Mistral-7B-Instruct-v0.2	0.54	0.47	0.65	0.47	80	41	44	91
gemma-1.1-2b-it	0.00	0.33	0.00	0.00	0	**85**	**0**	171
Random	0.33	0.33	0.33	0.33				

Table A.4: Performance of LLM annotations with **cropped** observations, subgoals **suggested** by the LLM, and **no token separator**.

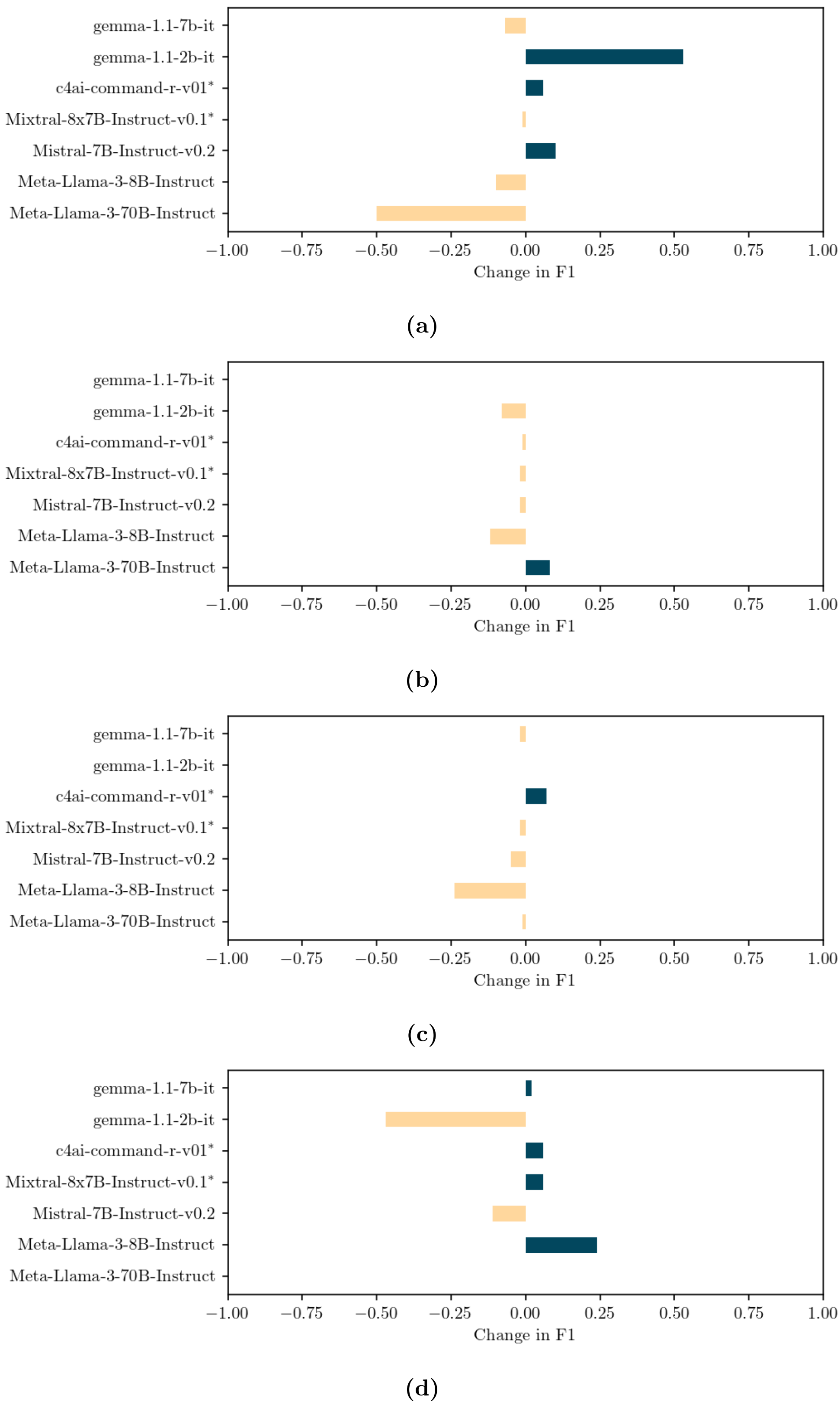
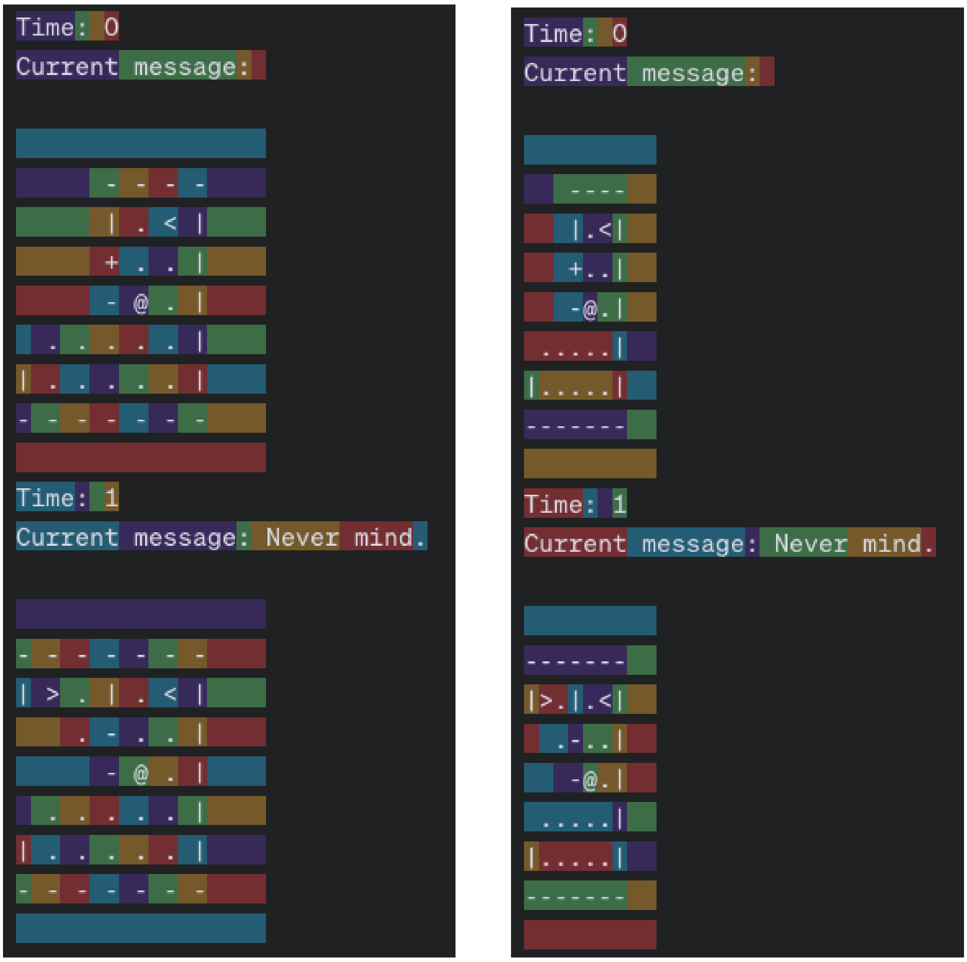


Figure A.2: Variation in F1 score between the baseline results presented in Tables 9.1-9.4 and the results without a token separator in Tables A.1-A.4. Yellow bars indicate worse performance without a separator. and blue otherwise.



(a) (b)

Figure A.3: Tokenisation of the same prompt, with **(a)** and without **(b)** a token separator (whitespace).

A.6.2 Actions

In this section we investigate the impacts of explicitly adding the action a_t to the transition (s_t, a_t, s_{t+1}), which was left implicit in the main experiments in Section 9.6.

Annotator	F1 ↑	Accuracy ↑	Precision ↑	Recall ↑	TP ↑	TN ↑	FP ↓	FN ↓
Human	**1.00**	**1.00**	**1.00**	**1.00**	**171**	**85**	**0**	**0**
Mixtral-8x7B-Instruct-v0.1	**0.83**	**0.79**	0.89	**0.77**	**132**	69	16	**39**
Mistral-7B-Instruct-v0.2	0.67	0.66	0.99	0.50	86	84	1	85
gemma-1.1-7b-it	0.66	0.66	0.97	0.50	86	82	3	85
c4ai-command-r-v01	0.66	0.64	0.93	0.51	87	78	7	84
Meta-Llama-3-8B-Instruct	0.61	0.62	0.97	0.44	76	83	2	95
Meta-Llama-3-70B-Instruct	0.46	0.53	0.98	0.30	51	84	1	120
gemma-1.1-2b-it	0.00	0.33	0.00	0.00	0	**85**	**0**	171

Table A.5: Performance of LLM annotations with **game screen** observations, subgoals **provided** in the prompt, and the transition includes actions.

Annotator	F1 ↑	Accuracy ↑	Precision ↑	Recall ↑	TP ↑	TN ↑	FP ↓	FN ↓
Human	**1.00**	**1.00**	**1.00**	**1.00**	**171**	**85**	**0**	**0**
Mixtral-8x7B-Instruct-v0.1	**0.80**	**0.75**	0.86	**0.75**	**128**	64	21	**43**
gemma-1.1-7b-it	0.75	0.68	0.79	0.72	123	52	33	48
c4ai-command-r-v01	0.72	0.68	0.86	0.62	106	68	17	65
gemma-1.1-2b-it	0.71	0.62	0.73	0.69	118	42	43	53
Mistral-7B-Instruct-v0.2	0.66	0.66	0.98	0.50	86	83	2	85
Meta-Llama-3-70B-Instruct	0.64	0.59	0.78	0.54	92	59	26	79
Meta-Llama-3-8B-Instruct	0.50	0.54	0.90	0.35	60	78	7	111

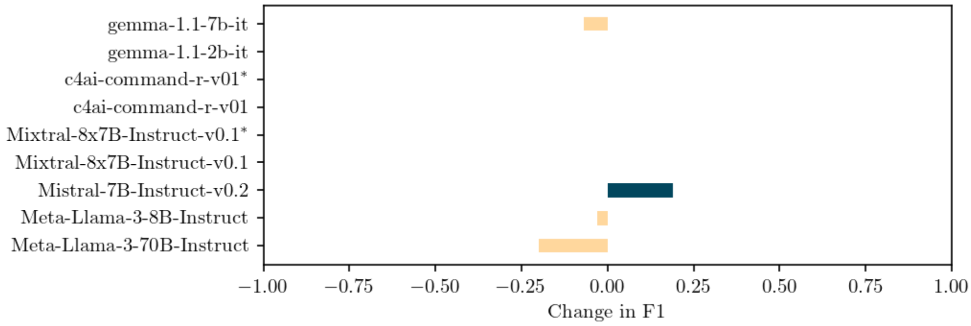
Table A.6: Performance of LLM annotations with **cropped** observations, subgoals **provided** in the prompt, and the transition includes actions.

Annotator	F1 ↑	Accuracy ↑	Precision ↑	Recall ↑	TP ↑	TN ↑	FP ↓	FN ↓
Human	**1.00**	**1.00**	**1.00**	**1.00**	**171**	**85**	**0**	**0**
Meta-Llama-3-70B-Instruct	**0.85**	**0.79**	0.80	**0.91**	**155**	46	39	**16**
Mixtral-8x7B-Instruct-v0.1	0.79	0.69	0.72	0.86	147	29	56	24
Meta-Llama-3-8B-Instruct	0.78	0.66	0.68	**0.91**	**155**	13	72	**16**
gemma-1.1-7b-it	0.75	0.64	0.69	0.82	141	22	63	30
Mistral-7B-Instruct-v0.2	0.71	0.61	0.70	0.71	122	33	52	49
c4ai-command-r-v01	0.71	0.67	**0.86**	0.60	102	69	16	69
gemma-1.1-2b-it	0.00	0.33	0.00	0.00	0	**85**	**0**	171

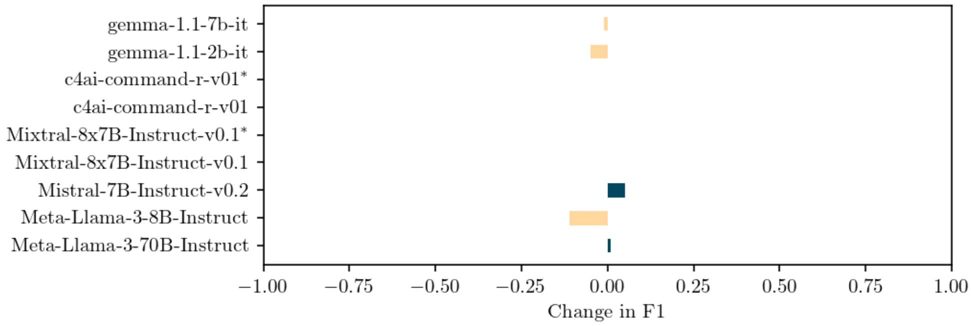
Table A.7: Performance of LLM annotations with **game screen** observations, subgoals **suggested** by the LLM, and the transition includes actions.

Annotator	F1 ↑	Accuracy ↑	Precision ↑	Recall ↑	TP ↑	TN ↑	FP ↓	FN ↓
Human	**1.00**	**1.00**	**1.00**	**1.00**	**171**	**85**	**0**	**0**
Meta-Llama-3-70B-Instruct	**0.86**	**0.80**	0.82	0.91	155	50	35	16
Mixtral-8x7B-Instruct-v0.1	0.83	0.75	0.76	0.91	156	35	50	15
gemma-1.1-7b-it	0.82	0.72	0.73	**0.92**	**158**	27	58	**13**
c4ai-command-r-v01	0.74	0.64	0.72	0.76	130	35	50	41
Mistral-7B-Instruct-v0.2	0.72	0.63	0.74	0.70	120	42	43	51
Meta-Llama-3-8B-Instruct	0.53	0.46	0.63	0.46	79	39	46	92
gemma-1.1-2b-it	0.42	0.48	0.83	0.28	48	**75**	**10**	123

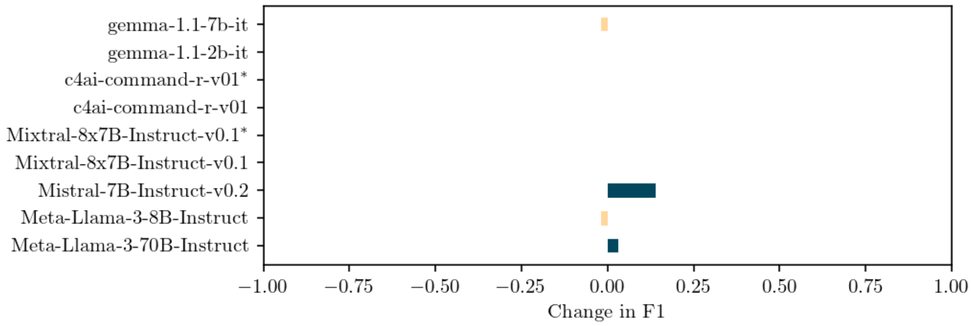
Table A.8: Performance of LLM annotations with **cropped** observations, subgoals **suggested** by the LLM, and the transition includes actions.



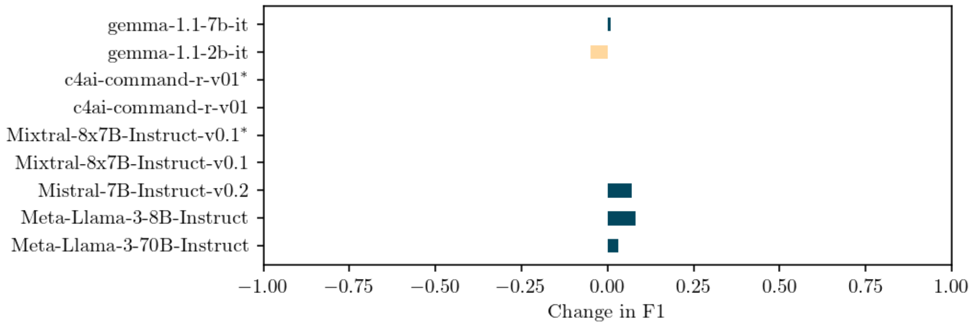
(a)



(b)



(c)



(d)

Figure A.4: Variation in F1 score between the baseline results presented in Tables 9.1-9.4 and the results where prompts also include the action in Tables A.5-A.8.

A.7 Details on NAVIX systems

Systems are *functions* that operate on the collective state of all entities, defining the rules of the interactions between them. In designing NAVIX, we aimed to maintain a bijective relationship between the systems and their respective mathematical formalism in RL. This makes it easier to translate the mathematical formalism into code, and vice versa, connecting the implementation to the theory. NAVIX includes the following systems: 1. `Intervention`: a function that updates the state of the entities according to the actions taken by the agents. 2. `Transition`: a function that updates the state of the entities according to the MDP state transitions. 3. `Observation`: a function that generates the observations that the agents receive. 4. `Reward`: a function that computes the rewards that the agents receive. 5. `Termination`: a function that determines if the episode is terminated. We now describe the systems formally.

The intervention is a function $I : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ that updates the state of the entities according to the actions taken by the agents. This corresponds to the canonical decision in an MDP.

The transition is a function $\mu : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ that updates the state of the entities according to the MDP state transitions. This corresponds to the canonical state transition kernel in an MDP.

The observation is a function $O : \mathcal{S} \to \mathcal{O}$ that generates the observations that the agents receive. NAVIX includes multiple observation functions, each generating a different type of observation, for example, a first-person view, a top-down view, or a third-person view, both in symbolic and pixel format. We provide both full and partial observations, allowing to cast a NAVIX environment both as an MDP or as a POMDP, depending on the needs of the algorithm. This follows the design of the original MiniGrid suite.

The reward is a function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that computes the rewards that the agents receive. Likewise, the termination is a function $\gamma : \mathcal{S} \rightarrow \{0, 1\}$ that determines if the episode is terminated. We include both the reward and the termination functions necessary to reproduce all MiniGrid environments. Both these systems rely on the concept of *events*, representing a goal to achieve. An *event* is itself an entity signalling that a particular state of the environment has been reached. For example, it can indicate that the agent has reached a particular cell, has picked up a particular object, or that the agent performed a certain action in a particular state.

We provide a summary of the implemented systems in NAVIX in Tables A.9, A.10, and A.11 for the observation, reward, and termination systems, respectively.

A.8 Reusable patterns

Here we provide some useful patterns that users can reuse as-they-are or modify to suit their needs. In particular, we show how to jit the full interaction loop of a NAVIX environment in Code 2, and how to run multiple seeds in parallel in Code 3. Further examples, including how to jit a whole training loop with a JAX-based agent, and how to automate hyperparameter search, are available in the NAVIX documentation at `https://epignatelli.com/navix/examples/getting_started.html`.

A.8.1 Jitting full interaction loops

```
import navix as nx

# init a NAVIX environment
env = nx.make("Navix-KeyCorridorS6R3-v0")

# sample a starting state
timestep = env.reset(key)

# jitting the step function
step_env = jax.jit(env.step)

# unroll the environment for 1000 steps
timestep, _ = jax.lax.scan(
    lambda timestep, _: (unroll(timestep, i % 6), ()),
    timestep,
    (timestep, jnp.arange(1000))
)
```

Code 2: Example code to jit a `Navix-Empty-5x5-v0` environment.

A.8.2 Running multiple seeds in parallel

```
import navix as nx

env = nx.make("Navix-KeyCorridorS6R3-v0")

# define the run function
def run(key):
    def step(state, action):
        timestep, key = state
        key, subkey = jax.random.split(key)
        action = jax.random.randint(subkey, (), 0, env.action_space.n)
        return (env.step(timestep, action), key), ()


    # unroll the environment for 1000 steps
    timestep = env.reset(key)
    timestep, _ = jax.lax.scan(
        step,
        timestep,
        ((timestep, key) jnp.arange(1000)),
    )
    return timestep

seeds = jax.random.split(jax.random.PRNGKey(0), 1000)
batched_end_steps = jax.jit(jax.vmap(run))(seeds)
```

Code 3: Example code to jit a `Navix-Empty-5x5-v0` environment.

A.9 Customising NAVIX environments

NAVIX is designed to be highly customisable, allowing users to create new environments by combining existing entities and systems. In this section, we provide examples of how to customise NAVIX environments by using different *systems*.

```
import navix a nx

reward_fn = nx.rewards.compose(
    nx.rewards.on_goal_reached(),
    nx.rewards.on_lava_fall()
)

env = nx.make(
    "Navix-Empty-5x5-v0",
    reward_fn=reward_fn)
```

Code 4: Example code to create a `Navix-Empty-5x5-v0` environment with a custom reward function. See Table A.10 for a list of implemented reward functions.

```
import navix as nx

env = nx.make(
    "Navix-Empty-5x5-v0",
    observation_fn=nx.observations.rgb())
```

Code 5: Example code to create a `Navix-Empty-5x5-v0` environment with a custom observation function. See Table A.9 for a list of implemented observation functions.

```
import navix as nx

env = nx.make(
    "Navix-Empty-5x5-v0",
    termination_fn=nx.terminations.on_goal_reached())
```

Code 6: Example code to create a `Navix-Empty-5x5-v0` environment with a custom termination function. See Table A.11 for a list of implemented termination functions.

For example, to create a new environment where the agent has to reach a goal while avoiding lava, we can combine the `Goal` and `Lava` entities with the

```python
import jax, navix as nx

class CustomComponent(nx.Componnet):
    """My custom component."""

    custom_property: jax.Array = nx.components.field(shape=())
```

Code 7: Example code to extend NAVIX by creating a custom component. Notice that the property must have a type annotation and specify a shape.

`Reward` system, as shown in Code 4.

Alternatively, to use a different observation function, we can use the `Observation` system, as shown in Code 5.

Finally, to terminate the environment, for example, only when the agent reaches the goal, but not when it falls into the lava, we can use the `Termination` system, as shown in Code 6.

These examples can be extended to create more complex environments by combining different systems for the same environment configuration.

A.10 Extending NAVIX environments

NAVIX is designed to be easily extensible. Users can create new entities, components, systems, and full environments by implementing the necessary functions. In this section, we provide **templates** to extend NAVIX environments. In particular, Code 8 shows how to create a custom environment, Code 7 shows how to create a custom component, Code 9 shows how to create a custom entity, and Code 10 shows how to create custom systems.

```python
import jax, navix as nx

class CustomEnv(nx.Environment):
    def _reset(self, key: jax.Array) -> nx.Timestep:
        """Reset the environment."""
        # create your grid, place your entities, define your mission
        return timestep


nx.registry.register_env(
    "CustomEnv",
    lambda *args, **kwargs: CustomEnv.create(
        observation_fn=nx.observations.symbolic(),
        reward_fn=nx.rewards.on_goal_reached(),
        termination_fn=nx.terminations.on_goal_reached(),
    )
)
```

Code 8: Example code to extend NAVIX by creating a custom environment. The `_reset` function allows to generate a custom starting state, after which the environment will evolve according to the usual systems: intervention, transition, reward and termination functions. Notice that it is convenient to use the environment constructor `create` to automatically set non-orthogonal properties (e.g. observation space and observation function).

```python
import jax, navix as nx

class CustomEntity(nx.Entity, CustomComponent):
    """My custom entity."""

    @property
    def walkable(self) -> jax.Array:
        return jnp.broadcast_to(jnp.asarray(False), self.shape)

    @property
    def transparent(self) -> jax.Array:
        return jnp.broadcast_to(jnp.asarray(False), self.shape)

    @property
    def sprite(self) -> jax.Array:
        sprite = # the address of your sprite, e.g., SPRITES_REGISTRY[Entities.WALL]
        return jnp.broadcast_to(sprite[None], (*self.shape, *sprite.shape))

    @property
    def tag(self) -> jax.Array:
        entity_id = # the id of your entity, e.g., EntityIds.WALL
        return jnp.broadcast_to(entity_id, self.shape)
```

Code 9: Example code to extend NAVIX by creating a custom entity. Notice that four properties must be implemented: `walkable`, `transparent`, `sprite`, and `tag`.

```
import jax, navix as nx

def my_reward_function(state: nx.State, action: nx.Action, new_state: nx.State) -> jax.Array
    """My custom reward function."""
    # do stuff
    return reward  # f32[]

def my_termination_function(state: nx.State, action: nx.Action, new_state: nx.State) -> jax.
    """My custom termination function."""
    # do stuff
    return termination  # bool[]

def my_observation_function(state: nx.State) -> jax.Array:
    """My custom observation function."""
    # do stuff
    return observation  # f32[]

def my_intervention_function(state: nx.State, action: nx.Action) -> nx.State:
    """My custom intervention function."""
    # do stuff
    return new_state  # State

def my_transition_function(state: nx.State) -> nx.State:
    """My custom transition function."""
    # do stuff
    return new_state  # State
```

Code 10: Example code to extend NAVIX by creating custom systems.

A.11 Additional results on NAVIX speedup

As reported in Section 10.4.1, NAVIX provides a significant speedup compared to the original Minigrid implementation. These improvements are due to both the migration of the computation to the GPU via XLA, which optimises the computation graph for the specific accelerator, and the batching of the environments. In Figure 10.3 we have shown the overall speedup. Here we provide an ablation study to understand the contribution of the batching and the parallel environments to the overall speedup. Figure A.5 shows the speedup of NAVIX compared to the original Minigrid implementation without batching, that is when only one environment is used. This is a suboptimal and unusual case in many RL training settings, where instead the tendency is to use multiple environments in parallel. We can observe that, while the results heavily depend on the environment and most of the NAVIX environments are still faster, the biggest contribution for the speedup is due to efficient batching.

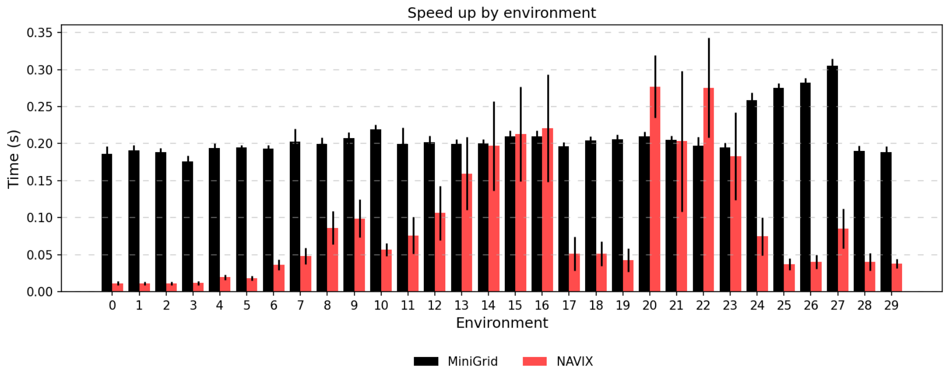


Figure A.5: Ablation. Speedup of NAVIX compared to the original Minigrid implementation without batching. The identifiers on the x-axis correspond to the environments as reported in Table A.12. Lower is better.

Observation function	Shape	Description
`symbolic`	`i32[H, W, 3]`	*The canonical grid encoding observation from MiniGrid.*
`symbolic_first_person`	`i32[R, R, 3]`	*A first-person view of the environment in symbolic format.*
`rgb`	`u8[32 * H, 32 * W, 3]`	*A fully visible image of the environment in RGB format.*
`rgb_first_person`	`u8[32 * R, 32 * R, 3]`	*A first-person view of the environment in RGB format.*
`categorical`	`i32[H, W]`	*A grid of entities ID in the environment.*
`categorical_first_person`	`i32[R, R]`	*A first-person view of the grid of entities ID.*

Table A.9: Implemented observation functions in NAVIX.

Reward function	Description
`on_goal_reached`	$+1$ *when a Goal entity and a Player entity have the same position*
`on_lava_fall`	-1 *when a Lava entity and a Player entoty have the same position*
`on_door_done`	$+1$ *when the `done` action is performed in front of a door with the colour specific in the `mission`*
`free`	0 everywhere
`action_cost`	$-cost_a$ *at every action taken, except `done`*
`time_cost`	$-cost_t$ *at every step*

Table A.10: Implemented reward functions in NAVIX.

Termination function	Description
`on_goal_reached`	*Terminates when a Goal entity and a Player entity have the same position*
`on_lava_fall`	*Terminates when a Lava entity and a Player entity have the same position*
`on_door_done`	*Terminates when the `done` action is performed in front of a door with the colour specific in the `mission`*
`free`	0 everywhere

Table A.11: Implemented termination functions in NAVIX.

A.12 Additional Tables

X tick	Env id
0	Navix-Empty-5x5-v0
1	Navix-Empty-6x6-v0
2	Navix-Empty-8x8-v0
3	Navix-Empty-16x16-v0
4	Navix-Empty-Random-5x5
5	Navix-Empty-Random-6x6
6	Navix-DoorKey-5x5-v0
7	Navix-DoorKey-6x6-v0
8	Navix-DoorKey-8x8-v0
9	Navix-DoorKey-16x16-v0
10	Navix-FourRooms-v0
11	Navix-KeyCorridorS3R1-v0
12	Navix-KeyCorridorS3R2-v0
13	Navix-KeyCorridorS3R3-v0
14	Navix-KeyCorridorS4R3-v0
15	Navix-KeyCorridorS5R3-v0
16	Navix-KeyCorridorS6R3-v0
17	Navix-LavaGapS5-v0
18	Navix-LavaGapS6-v0
19	Navix-LavaGapS7-v0
20	Navix-SimpleCrossingS9N1-v0
21	Navix-SimpleCrossingS9N2-v0
22	Navix-SimpleCrossingS9N3-v0
23	Navix-SimpleCrossingS11N5-v0
24	Navix-Dynamic-Obstacles-5x5
25	Navix-Dynamic-Obstacles-6x6
26	Navix-Dynamic-Obstacles-8x8
27	Navix-Dynamic-Obstacles-16x16
28	Navix-DistShift1-v0
29	Navix-DistShift2-v0

Table A.12: Correspondence between the x-ticks in Figure 10.3 and the environment ids.

Env-id	**Class**	**Height**	**Width**	**Reward**
Navix-Empty-5x5-v0	Empty	5	5	R_1
Navix-Empty-6x6-v0	Empty	6	5	R_1
Navix-Empty-8x8-v0	Empty	8	8	R_1
Navix-Empty-16x16-v0	Empty	16	16	R_1
Navix-Empty-Random-5x5	Empty	5	5	R_1
Navix-Empty-Random-6x6	Empty	6	6	R_1
Navix-Empty-Random-8x8	Empty	8	8	R_1
Navix-Empty-Random-16x16	Empty	16	16	R_1
Navix-DoorKey-5x5-v0	DoorKey	5	5	R_1
Navix-DoorKey-6x6-v0	DoorKey	6	6	R_1
Navix-DoorKey-8x8-v0	DoorKey	8	8	R_1
Navix-DoorKey-16x16-v0	DoorKey	16	16	R_1
Navix-DoorKey-Random-5x5	DoorKey	5	5	R_1
Navix-DoorKey-Random-6x6	DoorKey	6	6	R_1
Navix-DoorKey-Random-8x8	DoorKey	8	8	R_1
Navix-DoorKey-Random-16x16	DoorKey	16	16	R_1
Navix-FourRooms-v0	FourRooms	17	17	R_1
Navix-KeyCorridorS3R1-v0	KeyCorridor	3	7	R_1
Navix-KeyCorridorS3R2-v0	KeyCorridor	5	7	R_1
Navix-KeyCorridorS3R3-v0	KeyCorridor	7	7	R_1
Navix-KeyCorridorS4R3-v0	KeyCorridor	10	10	R_1
Navix-KeyCorridorS5R3-v0	KeyCorridor	13	13	R_1
Navix-KeyCorridorS6R3-v0	KeyCorridor	16	16	R_1
Navix-LavaGap-S5-v0	LavaGap	5	5	R_2
Navix-LavaGap-S6-v0	LavaGap	6	6	R_2
Navix-LavaGap-S7-v0	LavaGap	7	7	R_2
Navix-Crossings-S9N1-v0	Crossings	9	9	R_2
Navix-Crossings-S9N2-v0	Crossings	9	9	R_2
Navix-Crossings-S9N3-v0	Crossings	9	9	R_2
Navix-Crossings-S11N5-v0	Crossings	11	11	R_2
Navix-Dynamic-Obstacles-5x5	Dynamic-Obstacles	5	5	R_3
Navix-Dynamic-Obstacles-5x5	Dynamic-Obstacles	5	5	R_3
Navix-Dynamic-Obstacles-6x6	Dynamic-Obstacles	6	6	R_3
Navix-Dynamic-Obstacles-6x6	Dynamic-Obstacles	6	6	R_3
Navix-Dynamic-Obstacles-8x8	Dynamic-Obstacles	8	8	R_3
Navix-Dynamic-Obstacles-16x16	Dynamic-Obstacles	16	16	R_3
Navix-DistShift1-v0	DistShift	6	6	R_2
Navix-DistShift2-v0	DistShift	8	8	R_2
Navix-GoToDoor-5x5-v0	GoToDoor	5	5	R_1
Navix-GoToDoor-6x6-v0	GoToDoor	6	6	R_1
Navix-GoToDoor-8x8-v0	GoToDoor	8	8	R_1

Table A.13: List of environments available in NAVIX. *Env-id* denotes the id to instantiate the environment. Here, R_1 is the reward function for goal achievement – 1 when the agent is on the green square, and 0 otherwise. R_2 is the reward function for goal achievement and lava avoidance, adding -1 when the agent is on the lava square. R_3 is the reward function for goal achievement and dynamic obstacles avoidance, adding -1 when the agent is hit by a flying object. All environments terminate when the reward is not 0, for example, on goal achievement, or on lava collision.

A.13 Additional details on baselines

Algorithm	Fitted hyperparameters
PPO	#envs, #steps, #epochs, #minibatches, discount factor, λ (GAE), grad. norm clip, norm. obs., activation function
DQN	batch size, target network update freq., discount factor, exploration fraction, final ϵ, grad. norm clip, norm. obs., activation function
SAC	batch size, discount factor, τ (Polyak update), target entropy ratio, norm. obs., activation function

Table A.14: Fitted hyperparameters for PPO, DQN, and SAC.

Details on each hyperparameter set, for each environment and each algorithm are available at `https://github.com/keraJLi/rejax/tree/main/configs`.